

# macromedia® **DREAMWEAVER™3**

## Extending Dreamweaver



## Trademarks

Afterburner, AppletAce, Attain, Attain Enterprise Learning System, Attain Essentials, Attain Objects for Dreamweaver, Authorware, Authorware Attain, Authorware Interactive Studio, Authorware Star, Authorware Synergy, Backstage, Backstage Designer, Backstage Desktop Studio, Backstage Enterprise Studio, Backstage Internet Studio, Design in Motion, Director, Director Multimedia Studio, Doc Around the Clock, Dreamweaver, Dreamweaver Attain, Drumbeat, Drumbeat 2000, Extreme 3D, Fireworks, Flash, Fontographer, FreeHand, FreeHand Graphics Studio, Generator, Generator Developer's Studio, Generator Dynamic Graphics Server, Knowledge Objects, Knowledge Stream, Knowledge Track, Lingo, Live Effects, Macromedia, Macromedia M Logo & Design, Macromedia Flash, Macromedia Xres, Macromind, Macromind Action, MAGIC, Mediamaker, Object Authoring, Power Applets, Priority Access, Roundtrip HTML, Scriptlets, SoundEdit, ShockRave, Shockmachine, Shockwave, Shockwave Remote, Shockwave Internet Studio, Showcase, Tools to Power Your Ideas, Universal Media, Virtuoso, Web Design 101, Whirlwind and Xtra are trademarks of Macromedia, Inc. and may be registered in the United States or in other jurisdictions including internationally. Other product names, logos, designs, titles, words or phrases mentioned within this publication may be trademarks, servicemarks, or tradenames of Macromedia, Inc. or other entities and may be registered in certain jurisdictions including internationally.

This guide contains links to third-party Web sites that are not under the control of Macromedia, and Macromedia is not responsible for the content on any linked site. If you access a third-party Web site mentioned in this guide, then you do so at your own risk. Macromedia provides these links only as a convenience, and the inclusion of the link does not imply that Macromedia endorses or accepts any responsibility for the content on those third-party sites.

## Apple Disclaimer

APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED COMPUTER SOFTWARE PACKAGE, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE WHICH VARY FROM STATE TO STATE.

Copyright © 1999 Macromedia, Inc. All rights reserved. This manual may not be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without prior written approval of Macromedia, Inc.  
Part Number ZDW30M200

## Acknowledgments

Project Management: Sheila McGinn

Writing: Lori Hylan

Print and Help Production: Chris Basmajian

Special thanks to Heidi Bauer, Karen Catlin, Rob Christensen, Margaret Dumas, Jean Fitzgerald, Karen Gee, Jed Hartman, Steven Johnson, Sho Kuwamoto, Jay London, Joe Marini, Lisa Miller, Bob Tartar, Yoko Vogt, and the entire Dreamweaver team.

First Edition: December 1999

Macromedia, Inc.  
600 Townsend St.  
San Francisco, CA 94103

# CONTENTS

## CHAPTER 1

Extending Dreamweaver Overview ..... 7

Prerequisites .....8

Errata .....8

## CHAPTER 2

The Document Object Model ..... 9

The document object model in Dreamweaver .....10

How JavaScript works in extensions .....18

## CHAPTER 3

The Dreamweaver JavaScript API ..... 19

The most important function .....21

Behavior functions .....22

Clipboard functions .....33

Command functions .....37

Conversion functions .....38

CSS style functions .....39

External application functions.....44

File manipulation functions.....48

Find/replace functions.....55

Frame and frameset functions .....60

General editing functions .....61

Global application functions .....73

Global document functions.....74

History functions .....76

HTML style functions .....82

Keyboard functions .....85

Layer and image map functions.....91

Library and template functions .....93

Menu functions. . . . .	99
Path functions. . . . .	101
Quick Tag Editor functions. . . . .	103
Selection functions . . . . .	105
Site functions . . . . .	109
String manipulation functions. . . . .	123
Table editing functions . . . . .	125
Timeline functions . . . . .	130
Toggle functions . . . . .	135
Translation functions. . . . .	144
Visual layout functions . . . . .	146
Window functions . . . . .	150
Deprecated functions . . . . .	153
Enablers . . . . .	158
 <b>CHAPTER 4</b>	
The File I/O API. . . . .	177
Verifying that DWfile is installed . . . . .	177
The file I/O API . . . . .	178
 <b>CHAPTER 5</b>	
The Design Notes API. . . . .	183
How Design Notes work. . . . .	184
The Design Notes JavaScript API . . . . .	184
The Design Notes C API . . . . .	188
 <b>CHAPTER 6</b>	
The Fireworks Integration API . . . . .	193
The Fireworks integration API . . . . .	193
A simple Fireworks integration example . . . . .	198
 <b>CHAPTER 7</b>	
The HTTP API. . . . .	201
 <b>CHAPTER 8</b>	
C-Level Extensibility . . . . .	209

## **CHAPTER 9**

### **Objects** ..... 221

How object files work .....222

The object API .....222

Adding objects to the Object palette .....224

Adding objects to the Insert menu. ....225

## **CHAPTER 10**

### **Commands** .....227

How commands work .....228

The command API .....229

A simple command example .....232

Adding commands to the Commands menu .....233

## **CHAPTER 11**

### **Menu Commands** ..... 235

How menu commands work .....236

The menu command API .....237

A simple menu command .....240

A simple dynamic menu .....242

## **CHAPTER 12**

### **Property Inspectors** ..... 245

How property inspector files work. ....246

The property inspector API. ....247

A simple property inspector example. ....249

## **CHAPTER 13**

### **Floating Palettes** ..... 253

How floating palette files work .....253

The floating palette API .....255

About performance .....257

A simple floating palette example .....259

## **CHAPTER 14**

### **Behaviors** ..... 261

How behaviors work .....262

The behavior API .....263

A simple behavior example .....270

<b>CHAPTER 15</b>	
Data Translators .....	273
How data translators work .....	274
The data translator API .....	275
Determining what kind of translator to use .....	277
Adding a translated attribute to a tag .....	278
Locking translated tags or blocks of code .....	284
Finding bugs in your translator .....	293
<b>INDEX</b> .....	295

# CHAPTER 1

## Extending Dreamweaver Overview

---

You can extend Dreamweaver several ways using HTML, JavaScript and C. You can write your own objects, behavior actions, commands, property inspectors, floating palettes, data translators, and C libraries that can be called from within JavaScript code. To accomplish these tasks, you must be proficient in JavaScript and thoroughly understand HTML elements. You don't need to be proficient in C unless you need some functionality that is not available through the Dreamweaver JavaScript API.

This manual describes the Document Object Model (DOM) that Dreamweaver supports and the Dreamweaver JavaScript API—the custom JavaScript functions that are built into Dreamweaver and that can be used in any object, behavior action, command, property inspector, floating palette, or data translation file. The manual also describes the APIs that are specific to each type of file, and provides several commented code examples.

Throughout this book, object, behavior action, command, property inspector, floating palette, and data translation files are referred to collectively as *extensions*.

## Prerequisites

Because Dreamweaver extensions must be written in JavaScript, this manual assumes that readers are familiar with JavaScript syntax and basic programming concepts such as functions, variables, statements, operators, conditionals, and loops. This manual does not attempt to teach programming in general or JavaScript in particular.

If you are not yet an advanced JavaScript developer, but are familiar with the syntax of the language, start with objects to learn about extensibility. First read “The Document Object Model” on page 9, and then see “C-Level Extensibility” on page 209 for more information.

Anyone who wants to extend Dreamweaver should have a good JavaScript reference to help with syntax questions (for example, is it `substring()` or `subString()`?). Useful JavaScript references include *Javascript Bible* by Danny Goodman (IDG), *JavaScript: The Definitive Guide* by David Flanagan (O'Reilly), and *Pure JavaScript* by R. Allen Wyke, Jason D. Gilliam, and Charlton Ting (Sams). For a free JavaScript reference, see Netscape's DevEdge Online website at <http://developer.netscape.com:80/docs/manuals/javascript.html>.

## Errata

This documentation was written before the code in Dreamweaver 3 was complete. Thus, there may be some discrepancies between the final implementation of the JavaScript API in Dreamweaver 3 and how it is documented in this book. A list of known issues can be found in the Extensibility section of the Dreamweaver Support Center at <http://www.macromedia.com/support/dreamweaver/extend.html>.



## CHAPTER 2

### The Document Object Model

---

HTML documents consist of a tree of tags that reveals the document's structure. The root of the tree is the HTML tag. The two largest branches of the tree are HEAD and BODY. Offshoots of HEAD include TITLE, STYLE, SCRIPT, ISINDEX, BASE, META, and LINK. Offshoots of BODY include headings (H1, H2, and so on), block-level elements (P, DIV, FORM, and so on), text-level elements (FONT, BR, IMG, and so on), and the ADDRESS element. Leaves on the offshoots include attributes such as WIDTH, HEIGHT, ALT, and HREF.

A document object model, or DOM, is also a tree that discloses the document's structure. The DOM, however, reports this structure in terms of objects and properties, rather than in terms of tags and attributes.

The root of the DOM tree is the document itself, the HTML object is the trunk, and the rest of the objects in the document branch from the HTML object as the HTML tags and attributes do.

## The document object model in Dreamweaver

A browser's DOM determines how the JavaScript in an HTML document works in that browser. Similarly, Dreamweaver's DOM determines how the JavaScript in extensions works in Dreamweaver.

Dreamweaver's DOM combines a subset of the Netscape Navigator 4.0 DOM with a subset of the World Wide Web Consortium (W3C)'s DOM Level 1. With the incorporation of DOM Level 1, every part of an HTML page has become an object—including tags (which the W3C calls elements), comments, and text.

Objects can be referred to by index (`document.forms[3].elements[1]`) or by name (`document.myForm.myButton`). Objects with the same name are collapsed into an array. You can access a particular object in the group by index (for example, the first radio button with the name `myRadioGroup` in `myForm` would be referenced as `document.myForm.myRadioGroup[0]`).

The following table gives an overview of the properties, methods, and events supported by each object; these are described in more detail in books such as *JavaScript: The Definitive Guide* (O'Reilly). Additional details about the W3C properties and methods, which are less thoroughly documented by third parties, follow the table. A bullet (•) marks read-only properties.

Object	Properties	Methods	Events
window	<code>document</code> • <code>navigator</code> • <code>innerWidth</code> • <code>innerHeight</code> • <code>screenX</code> • <code>screenY</code> •	<code>alert()</code> <code>confirm()</code> <code>escape()</code> <code>unescape()</code> <code>close()</code> <code>setTimeout()</code> <code>clearTimeout()</code> <code>setInterval()</code> <code>clearInterval()</code> <code>resizeTo()</code>	<code>onResize</code>
navigator	<code>platform</code> •	None	None

Object	Properties	Methods	Events
document	forms • (an array of form objects) images • (an array of image objects) layers • (an array of LAYER, ILAYER, and absolutely positioned DIV and SPAN objects) <i>child objects by name</i> • nodeType • parentNode • childNodes • documentElement • body • URL • parentWindow •	getElementsByTagName() hasChildNodes()	onLoad
all tags/elements	nodeType • parentNode • childNodes • tagName • <i>attributes by name</i> innerHTML outerHTML	getAttribute() setAttribute() removeAttribute() getElementsByTagName() hasChildNodes()	
form	In addition to the properties available for all tags: elements • (an array of button, checkbox, password, radio, reset, select, submit, text, file, hidden, image, and textarea objects) <i>child objects by name</i> •	Only those methods available to all tags.	None
layer	In addition to the properties available for all tags: visibility left top width height zIndex	Only those methods available to all tags.	None

Object	Properties	Methods	Events
image	In addition to the properties available for all tags: src	Only those methods available to all tags.	onMouseOver onMouseOut onMouseDown onMouseUp
button reset submit	In addition to the properties available for all tags: form •	In addition to the methods available for all tags: blur() focus()	onClick
checkbox radio	In addition to the properties available for all tags: checked form •	In addition to the methods available for all tags: blur() focus()	onClick
password text file hidden image (field) textarea	In addition to the properties available for all tags: form • value	In addition to the methods available for all tags: blur() focus() select()	onBlur onFocus
select	In addition to the properties available for all tags: form • options • (an array of option objects) selectedIndex	In addition to the methods available for all tags: blur() (Windows only) focus() (Windows only)	onBlur (Windows only) onChange onFocus (Windows only)
option	In addition to the properties available for all tags: text	Only those methods available to all tags.	None
array boolean date function math number object string regexp	Matches Netscape 4	Matches Netscape 4	None

Object	Properties	Methods	Events
text	nodeType • parentNode • childNodes • data	hasChildNodes()	None
comment	nodeType • parentNode • childNodes • data	hasChildNodes()	None
odelist	length •	item()	None

While Dreamweaver has a DOM that resembles that of a browser, the property inspectors, floating palettes, and parameters and options dialog boxes associated with extensions are not browsers. For this reason, links (A tags) are not supported. In addition, while live plugins (set to play at all times) are supported in the BODY of extensions, Java applets and ActiveX controls are not.

## The dreamweaver object and its properties

Dreamweaver implements the standard objects as defined by the browsers and the W3C, as well as two custom objects: dreamweaver and site. The dreamweaver object has two read-only properties associated with it: appName and appVersion.

appName always has the value "Dreamweaver". appVersion has a value of the form "*versionNumber [languageCode] (platform)*". For example, the value of the appVersion property for the Swedish Windows version of Dreamweaver 3 would be "3.0 [se] (Win32)"; the value for the English Macintosh version would be "3.0 [en] (MacPPC)".

The appName and appVersion properties were implemented in Dreamweaver 3 and are not available in earlier versions of Dreamweaver. To determine whether the version of Dreamweaver is 3 or later, you can simply check for the existence of the appVersion or appName property. To check for a specific version of Dreamweaver, check first for the existence of appVersion and then for the version number. For example:

```
if (dreamweaver.appVersion && dreamweaver.appVersion.indexOf('3.01') != -1){
    // execute code
}
```

The site object has no properties. For more information about the methods of the dreamweaver and site objects, see "The Dreamweaver JavaScript API" on page 19.

## DOM details

Unlike the Netscape DOM, DOM Level 1 has not been documented in hundreds of third-party books and web sites. Thus, this document will describe the DOM Level 1 properties and methods, and the values they return, in some detail.

DOM Level 1 introduces four constants that describe the types of objects that make up the document tree (called *nodes*). These constants, which usually show up as return values for the `nodeType` property, are:

```
Node.DOCUMENT_NODE
Node.ELEMENT_NODE
Node.COMMENT_NODE
Node.TEXT_NODE
```

## Properties and methods of the document object

The following table lists the new properties and methods of the document object in Dreamweaver, along with their return values (with explanations, where appropriate). A bullet (•) marks read-only properties.

Property or method	Return value and explanation
<code>nodeType</code> •	<code>Node.DOCUMENT_NODE</code>
<code>parentNode</code> •	<code>null</code>
<code>parentWindow</code> •	The JavaScript object corresponding to the document's parent window. (This property is not included in DOM Level 1; however, it is supported by IE 4.0.)
<code>childNodes</code> •	A <code>nodeList</code> containing all the immediate children of the document object. Typically the document will have a single child: the HTML object.
<code>documentElement</code> •	The JavaScript object corresponding to the HTML tag. This property is shorthand for getting the value of <code>document.childNodes</code> and extracting the HTML tag from the <code>nodeList</code> .
<code>body</code> •	The JavaScript object corresponding to the BODY tag. This property is shorthand for calling <code>document.documentElement.childNodes</code> and extracting the BODY tag from the <code>nodeList</code> . For frameset documents, this property returns the node for the outermost frameset instead.
<code>URL</code> •	The <code>file://</code> URL for the document or, if the file has not been saved, an empty string.

Property or method	Return value and explanation
<code>getElementsByTagName(tagName)</code>	<p>A nodelist that can be used to step through tags of type <i>tagName</i> (for example, IMG, DIV, and so on). Provides similar functionality to the <code>dreamweaver.getObjectTags()</code> function.</p> <p>If the <i>tag</i> argument is LAYER, the function returns all LAYER and ILAYER tags and all absolutely positioned DIV and SPAN tags.</p> <p>If the <i>tag</i> argument is INPUT, the function returns all form elements. (For this shortcut to work properly, all form field names must begin with a letter.)</p>
<code>hasChildNodes()</code>	TRUE

## Properties and methods of HTML tag objects

Every HTML tag is represented by a JavaScript object. Tags are organized in a tree hierarchy, where tag *x* is a parent of tag *y* if *y* falls completely within *x*'s opening and closing tags (`<x>x content surrounding <y>y content</y></x>`). The following table lists the properties and methods of tag objects in Dreamweaver, along with their return values (with explanations, where appropriate). A bullet (•) marks read-only properties.

Property or method	Return value and explanation
<code>nodeType</code> •	Node.ELEMENT_NODE
<code>parentNode</code> •	The parent tag. If this is the HTML tag, then the document object is returned instead.
<code>childNodes</code> •	A nodelist containing all the immediate children of the tag.
<code>tagName</code> •	The HTML name for the tag, such as IMG, A, or BLINK. This value is always returned in uppercase letters.
<code>attrName</code>	A string containing the value of the specified tag attribute. <code>tag.attrName</code> cannot be used if <code>attrName</code> is a reserved word in the JavaScript language (for example, class). In this case, use <code>getAttribute()</code> and <code>setAttribute()</code> instead.
<code>innerHTML</code>	The HTML source contained between the begin tag and the end tag. For example, in the code <code>&lt;p&gt;&lt;b&gt;Hello&lt;/b&gt;, World!&lt;/p&gt;</code> , <code>p.innerHTML</code> would return <code>&lt;b&gt;Hello&lt;/b&gt;, World!</code> . If you write to this property, the DOM tree is immediately updated to reflect the new structure of the document. (This property is not included in DOM Level 1; however, it is supported by IE 4.0.)

Property or method	Return value and explanation
<code>outerHTML</code>	The HTML source for this tag, including the tag itself. For the example code above, <code>p.outerHTML</code> would return <code>&lt;p&gt;&lt;b&gt;Hello&lt;/b&gt;, World!&lt;/p&gt;</code> . If you write to this property, the DOM tree is immediately updated to reflect the new structure of the document. (This property is not included in DOM Level 1; however, it is supported by IE 4.0.)
<code>getAttribute(attrName)</code>	The value of the specified attribute if it is explicitly specified; otherwise, null.
<code>getTranslatedAttribute(attrName)</code>	The translated value of the specified attribute, or the same value that would be returned by <code>getAttribute()</code> if the attribute's value is not translated. (This property is not included in DOM Level 1; it was added to Dreamweaver 3 to support attribute translation.)
<code>setAttribute(attrName, attrValue)</code>	No return value. Sets the specified attribute to the specified value: for example, <code>img.setAttribute("src", "image/roses.gif")</code> .
<code>removeAttribute(attrName)</code>	No return value. Removes the specified attribute and its value from the HTML for this tag.
<code>getElementsByTagName(tagName)</code>	A nodelist that can be used to step through child tags of type <i>tagName</i> (for example, IMG, DIV, and so on). If the <i>tag</i> argument is LAYER, the function returns all LAYER and ILAYER tags and all absolutely positioned DIV and SPAN tags. If the <i>tag</i> argument is INPUT, the function returns all form elements. (For this shortcut to work properly, all form field names must begin with a letter.)
<code>hasChildNodes()</code>	A Boolean value indicating whether the tag has any children.
<code>hasTranslatedAttributes()</code>	A Boolean value indicating whether the tag has any translated attributes. (This property is not included in DOM Level 1; it was added to Dreamweaver 3 to support attribute translation.)



## Properties and methods of text objects

Each contiguous block of text in an HTML document (for example, the text within a P tag) is represented by a JavaScript object. Text objects never have children. The following table lists the properties and methods of text objects in Dreamweaver, along with explanations where appropriate. A bullet (•) marks read-only properties.

Property or method	Return value and explanation
nodeType •	Node.TEXT_NODE
parentNode •	The parent tag.
childNodes •	An empty nodelist.
data	The actual text string. Entities in the text are represented as a single character (for example, the text Joseph & I is returned as Joseph & I).
hasChildNodes()	FALSE

## Properties and methods of comment objects

Each HTML comment is represented by a JavaScript object. The following table lists the properties and methods of comment objects in Dreamweaver, along with explanations where appropriate. A bullet (•) marks read-only properties.

Property or method	Return value and explanation
nodeType •	Node.COMMENT_NODE
parentNode •	The parent tag.
childNodes •	An empty nodelist.
data	The text string between the comment markers (<!-- and -->).
hasChildNodes()	FALSE

## How JavaScript works in extensions

When Dreamweaver processes extensions, it compiles everything between `SCRIPT` tags and executes any code within `SCRIPT` tags that is not part of a function declaration (for example, initialization of global variables). It also reads in, compiles, and executes scripts in external JavaScript files specified in the `SRC` attributes of `SCRIPT` tags.

**Note:** If any JavaScript code in your extension files contains the string `'</SCRIPT>'`, the JavaScript interpreter reads this as an actual closing `SCRIPT` tag and reports an unterminated string literal error. To avoid this problem, break the string into pieces and concatenate them, like this: `'<' + '/SCRIPT>'`.

In command and behavior action files, Dreamweaver executes code in the `onLoad` event handler (if one appears in the `BODY` tag) when the user chooses the command or action from a menu. In object files, Dreamweaver executes code in the `onLoad` event handler on the `BODY` tag if the body of the document contains a form. Dreamweaver ignores the `onLoad` handler on the `BODY` tag in data translator, property inspector, and floating palette files. In all extensions, Dreamweaver executes code in other event handlers (for example, `onBlur="alert('This is a required field.')"`) when the user interacts with the form fields to which they are attached.

JavaScript URLs (for example, `<a href="javascript:alert('hi')">`) are not supported, nor is the `document.write()` statement.

### Running scripts at startup or shutdown

In Dreamweaver 3, if you place a command file in the `Configuration/Startup` folder, the command runs as Dreamweaver is starting up. Startup commands load before the `menus.xml` file, before the files in the `ThirdPartyTags` folder, and before any other commands, objects, behaviors, inspectors, floating palettes, or translators. Thus, you can use startup commands to modify the `menus.xml` file or other extension files. You can also show warnings or prompt the user for information, but you cannot call `dreamweaver.runCommand()`.

Similarly, if you place a command file in the `Configuration/Shutdown` folder, the command will run as Dreamweaver is shutting down. You can call `“dreamweaver.runCommand()”` on page 37 from shutdown commands, show warnings, or prompt the user for information, but you cannot stop the shutdown process.

For more information about commands, see “Commands” on page 227.

## CHAPTER 3

### The Dreamweaver JavaScript API

---

The objects, properties, and especially the methods described in “The Document Object Model” on page 9, were a good foundation on which to build extensibility into Dreamweaver. Several tasks unique to authoring environments cannot be accomplished with the methods available in the Netscape, Microsoft, or W3C DOMs, however. The most obvious example of this is selection, which is integral to the user experience in an authoring environment: DOM Level 1 and the proprietary browser DOMs do not address selection because users cannot select and modify content (except in form fields) in a browser window.

To make creating useful Dreamweaver extensions and customizing Dreamweaver menus possible, Dreamweaver exposes more than 400 JavaScript functions to developers beyond the standards-based DOM methods. This represents a huge expansion over what was available in Dreamweaver 2. Almost any task that the user can accomplish in Dreamweaver with the menus, floating palettes, inspectors, Site window, or Document window can now be done with JavaScript.

## Understanding the objects in the API

All of the custom functions in the following sections are methods of the dreamweaver object, the site object, or the object that represents a document's DOM. Methods that fit into the latter category are listed here as `dom.functionName()`. To produce the desired results, you must first get the DOM of a document and call the functions as methods of that DOM. You cannot simply type `dom.functionName()`. For example:

```
var currentDOM = dreamweaver.getDocumentDOM('document');
currentDOM.setSelection(100,200);
currentDOM.clipCopy();
var otherDOM = dreamweaver.openDocument(dreamweaver.getSiteRoot() + "html/
foo.htm");
otherDOM.endOfDocument();
otherDOM.clipPaste();
```

Unless otherwise noted, methods of the dom object can only operate on open documents, and running a function on a document that isn't open will cause Dreamweaver to report an error. Methods of the dom object that can operate only on the active document or on closed documents indicate this fact in their descriptions.

In Dreamweaver 3, dw is synonymous with dreamweaver. Thus all of the methods of the dreamweaver object can be referred to as `dw.functionName()`. This notation appears in code examples throughout this and following chapters.

## About enablers

Because every menu item in Dreamweaver 3 is implemented as a JavaScript function, a JavaScript mechanism is required to determine when menu items should be enabled. This mechanism is a series of functions called *enablers*.

An enabler determines whether its associated main function can operate in the current context. For example, `site.canGet()` determines whether Dreamweaver can perform a Get operation (`site.get()`).

Each function specification in this API lists the enabler for the function, if applicable. (Some functions do not have enablers, either because the menu item associated with the function is always enabled, or because the function is unrelated to menus.) Function specifications for the enablers appear at the end of this chapter.

## How this chapter is organized

The methods in the Dreamweaver JavaScript API are grouped functionally and then alphabetically by object and then by method name. For example, methods that deal with creating, applying, and deleting CSS styles are grouped under CSS Style functions; methods of the dom object are listed first, followed by methods of the dreamweaver object, followed by methods of the dreamweaver.cssStylePalette object. Optional arguments are enclosed in curly braces ({ }).

## The most important function

Virtually all of the functions involved in DOM manipulations require that you first determine which DOM to change. This is the task of `dreamweaver.getDocumentDOM()`, and thus it is the most important function.

### `dreamweaver.getDocumentDOM()`

<b>Availability</b>	2.0
<b>Description</b>	Provides access to the tree of objects for the specified document. After the tree of objects is returned to the caller, the caller can edit the tree to change the contents of the document.
<b>Arguments</b>	<p><i>sourceDoc</i></p> <p>The argument must be "document", "parent", "parent.frames[<i>number</i>]", "parent.frames[<i>frameName</i>]", or a URL. <i>document</i> specifies the document that has the focus and contains the current selection. <i>parent</i> specifies the parent frameset (if the currently selected document is in a frame), and <i>parent.frames[<i>number</i>]</i> and <i>parent.frames[<i>frameName</i>]</i> specify a document that is in a particular frame within the frameset containing the current document. If the argument is a relative URL, it is relative to the extension file. In Dreamweaver 3, <i>sourceDoc</i> defaults to <i>document</i> if omitted.</p> <p><b>Note:</b> If the argument is "document", the caller must be <code>applyBehavior()</code>, <code>deleteBehavior()</code>, <code>objectTag()</code>, or any function in a command or property inspector file in order to perform edits to the document.</p>
<b>Returns</b>	The JavaScript document object at the root of the tree.
<b>Enabler</b>	None.
<b>Example</b>	<p>The following code snippet uses <code>dreamweaver.getDocumentDOM()</code> to access the background color of the current document:</p> <pre>var theDOM = dreamweaver.getDocumentDOM("document"); theDOM.body.bgcolor = "#000000";</pre>

## Behavior functions

Behavior functions let you add behaviors to and remove behaviors from an object, find out which behaviors are attached to an object, get information about the object to which a behavior is attached, and so on. Methods of the `dreamweaver.behaviorInspector` object either control or act on the selection in the Behavior inspector, not in the current document.

### `dom.addBehavior()`

<b>Availability</b>	3.0
<b>Description</b>	Adds a new event/action pair to the selected element. This function is valid only for the active document.
<b>Arguments</b>	<p><i>event</i>, <i>action</i>, {<i>eventBasedIndex</i>}</p> <ul style="list-style-type: none"><li>• The first argument is the JavaScript event handler that should be used to attach the behavior to the element; for example, <code>onClick</code>, <code>onMouseOver</code>, or <code>onLoad</code>.</li><li>• The second argument is the function call that would be returned by <code>applyBehavior()</code> if the action were added using the Behavior inspector; for example, <code>"MM_popupMsg('Hello World')"</code>.</li><li>• The third argument is the position at which this action should be added. <i>eventBasedIndex</i> is a zero-based index; if two actions already are associated with the specified event, and you specify <i>eventBasedIndex</i> as 1, this action will be executed between the other two. If this argument is omitted, the action is added after all existing actions for the specified event.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.getBehavior()

<b>Availability</b>	3.0
<b>Description</b>	Gets the action at the specified position within the specified event. This function acts on the current selection and is valid only for the active document.
<b>Arguments</b>	<i>event</i> , { <i>eventBasedIndex</i> }
	<ul style="list-style-type: none"><li>• The first argument is the JavaScript event handler through which the action is attached to the element; for example, <code>onClick</code>, <code>onMouseOver</code>, or <code>onLoad</code>.</li><li>• The second argument is the position of the action to get. For example, if two actions are associated with the specified event, 0 is the first one and 1 is the second. If this argument is omitted, all the actions for the specified event are returned.</li></ul>
<b>Returns</b>	A string representing the function call (for example, <code>"MM_swapImage(document.Image1,'document.Image1','foo.gif','#933292969950')"</code> ), or an array of strings if <i>eventBasedIndex</i> was omitted.
<b>Enabler</b>	None.

## dom.reapplyBehaviors()

<b>Availability</b>	3.0
<b>Description</b>	Checks to make sure that the functions associated with any behavior calls on the specified node are in the HEAD of the document, and inserts them if they are missing.
<b>Arguments</b>	{ <i>elementNode</i> }
	The argument is a an element node within the current document. If the argument is omitted, Dreamweaver checks all element nodes in the document for orphaned behavior calls.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.removeBehavior()

<b>Availability</b>	3.0
<b>Description</b>	Removes the action at the specified position within the specified event. This function acts on the current selection and is valid only for the active document.
<b>Arguments</b>	<p><i>event</i>, {<i>eventBasedIndex</i>}</p> <ul style="list-style-type: none"><li>• The first argument is the event handler through which the action is attached to the element; for example, <code>onClick</code>, <code>onMouseOver</code>, or <code>onLoad</code>. If this argument is omitted, all actions are removed from the element.</li><li>• The second argument is the position of the action to be removed. For example, if two actions are associated with the specified event, 0 is the first one and 1 is the second. If this argument is omitted, all the actions for the specified event are removed.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.



## **dreamweaver.getBehaviorElement()**

<b>Availability</b>	2.0
<b>Description</b>	Gets the DOM object corresponding to the tag to which the behavior is being applied. This function is applicable only in behavior action files.
<b>Arguments</b>	None.
<b>Returns</b>	<p>A DOM object or null. This function returns null under the following circumstances:</p> <ul style="list-style-type: none"><li>• When the current script is not executing within the context of the Behavior inspector.</li><li>• When the Behavior inspector is being used to edit a behavior in a timeline.</li><li>• When the currently executing script was invoked by <code>dreamweaver.popupAction()</code>.</li><li>• When the Behavior inspector is attaching an event to a link wrapper and the link wrapper does not yet exist.</li><li>• When this function appears outside of an action file.</li></ul>
<b>Enabler</b>	None.
<b>Example</b>	<p><code>dreamweaver.getBehaviorElement()</code> can be used in the same way as <code>dreamweaver.getBehaviorTag()</code> to determine whether the selected action is appropriate for the selected HTML tag, except that it gives you access to more information about the tag and its attributes. For example, if you write an action that can be applied only to a hyperlink (A HREF) that does not target another frame or window, you can use <code>getBehaviorElement()</code> as part of the function that initializes the user interface for the parameters dialog box.</p> <pre>function initializeUI(){     var theTag = dreamweaver.getBehaviorElement();     var CANBEAPPLIED = (theTag.tagName == "A"         &amp;&amp; theTag.getAttribute("HREF") != null         &amp;&amp; theTag.getAttribute("TARGET") == null);     if (CANBEAPPLIED) {         // display the action UI     } else{         // display a helpful message that tells the user         // that this action can only be applied to a         // hyperlink without an explicit target]     } }</pre>

## **dreamweaver.getBehaviorTag()**

<b>Availability</b>	1.2
<b>Description</b>	Gets the source of the tag to which the behavior is being applied. This function is applicable only in action files.
<b>Arguments</b>	None.
<b>Returns</b>	A string representing the source of the tag. This is the same string that is passed as an argument ( <i>HTMLelement</i> ) to the <code>canAcceptBehavior()</code> function. If this function appears outside of an action file, the return value is an empty string.
<b>Enabler</b>	None.
<b>Example</b>	If you write an action that can be applied only to a hyperlink (A HREF), you can use <code>getBehaviorTag()</code> as part of the function that initializes the user interface for the parameters dialog box.

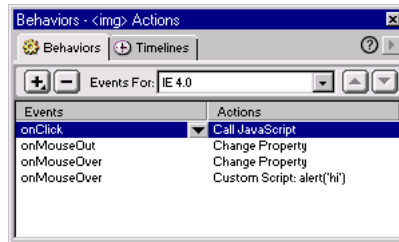
```
function initializeUI(){
    var theTag = dreamweaver.getBehaviorTag().toUpperCase();
    var CANBEAPPLIED = (theTag.indexOf('HREF') != -1);
    if (CANBEAPPLIED) {
        // display the action UI
    } else{
        // display a helpful message that tells the user
        // that this action can only be applied to a
        // hyperlink
    }
}
```

## **dreamweaver.popupAction()**

<b>Availability</b>	2.0
<b>Description</b>	<p>Presents the user with a parameters dialog box for the specified behavior action. To the user, the effect is the same as selecting the action from the Actions pop-up menu in the Behavior inspector. This function lets extension files other than actions attach behaviors to objects in the user's document. It blocks other edits until the user dismisses the dialog box.</p> <p><b>Note:</b> This function can be called only within <code>objectTag()</code> or in any script in a command or property inspector file.</p>
<b>Arguments</b>	<p><i>actionName</i>, {<i>funcCall</i>}</p> <ul style="list-style-type: none"><li>• The first argument is the name of a file in the Configuration/Behaviors/Actions folder that contains a JavaScript behavior action (for example, "Timeline/Play Timeline.htm").</li><li>• The second argument is a string containing a function call for the action specified in <i>actionName</i> (for example, "MM_playTimeline(...)"). This argument, if specified, is supplied by the <code>applyBehavior()</code> function in the action file.</li></ul>
<b>Returns</b>	<p>The function call for the behavior action. When the user clicks OK in the parameters dialog box, the behavior is added to the current document (the appropriate functions are added to the HEAD of the document, HTML may be added to the top of the BODY, and other edits may be made to the document). The function call (for example, "MM_playTimeline(...)") is not added to document, but becomes the return value of this function.</p>
<b>Enabler</b>	None.

## dreamweaver.behaviorInspector.getBehaviorAt()

<b>Availability</b>	3.0
<b>Description</b>	Gets the event/action pair at the specified position in the Behavior inspector.
<b>Arguments</b>	<i>positionIndex</i>
<b>Returns</b>	An array of two items: <ul style="list-style-type: none"><li>• An event handler</li><li>• A function call or JavaScript statement</li></ul>
<b>Enabler</b>	None.
<b>Example</b>	Because <i>positionIndex</i> is a zero-based index, if the Behavior inspector displays the list shown, a call to <code>dw.behaviorInspector.getBehaviorAt(2)</code> returns an array containing two strings: "onMouseOver" and "MM_changeProp('document.moon','document.moon','src','sun.gif','IMG')".



## dreamweaver.behaviorInspector.getBehaviorCount()

<b>Availability</b>	3.0
<b>Description</b>	Counts the number of actions attached to the currently selected element through event handlers.
<b>Arguments</b>	None.
<b>Returns</b>	An integer representing the number of actions attached to the element. This number is equivalent to the number of actions visible in the Behavior inspector and includes both Dreamweaver behavior actions and custom JavaScript.
<b>Enabler</b>	None.
<b>Example</b>	A call to <code>dw.behaviorInspector.getBehaviorCount()</code> for the selected link <code>&lt;A HREF="javascript:setCookie()" onClick="MM_popupMsg('A cookie has been set.');"parent.rightframe.location.href='aftercookie.html'"&gt;</code> would return 2.

## dreamweaver.behaviorInspector.getSelectedBehavior()

**Availability** 3.0

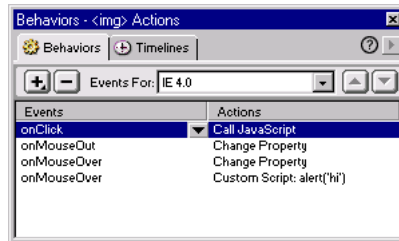
**Description** Gets the position in the Behavior inspector of the selected action.

**Arguments** None.

**Returns** An integer representing the position in the Behavior inspector of the selected action, or -1 if no action is selected.

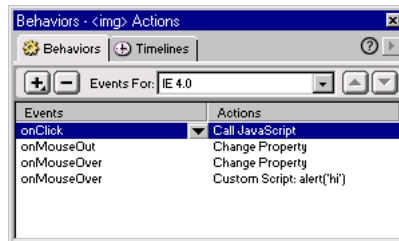
**Enabler** None.

**Example** If the first action in the Behavior inspector is selected, as shown, a call to `dw.behaviorInspector.getSelectedBehavior()` returns 0.



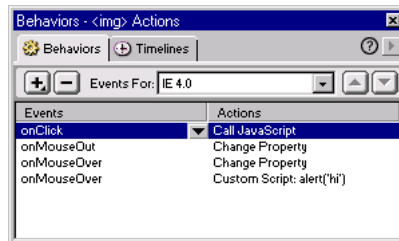
## `dreamweaver.behaviorInspector.moveBehaviorDown()`

<b>Availability</b>	3.0
<b>Description</b>	Moves a behavior action lower in sequence by changing its execution order within the scope of an event.
<b>Arguments</b>	<i>positionIndex</i>  The argument is the position of the action in the Behavior inspector. The first action in the list is at position 0.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	Assuming the Behavior inspector setup shown in this example, calling <code>dw.behaviorInspector.moveBehaviorDown(2)</code> would swap the positions of the Custom Script and the Change Property actions on the <code>onMouseOver</code> event. Calling <code>dw.behaviorInspector.moveBehaviorDown()</code> for any other position would have no effect because the <code>onClick</code> and <code>onMouseOut</code> events each have only one behavior associated with them, and the behavior at position 3 is already at the bottom of the <code>onMouseOver</code> group.



## `dreamweaver.behaviorInspector.moveBehaviorUp()`

<b>Availability</b>	3.0
<b>Description</b>	Moves a behavior higher in sequence by changing its execution order within the scope of an event.
<b>Arguments</b>	<i>positionIndex</i>  The argument is the position of the action in the Behavior inspector. The first action in the list is at position 0.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	Assuming the Behavior inspector setup shown in this example, calling <code>dw.behaviorInspector.moveBehaviorUp(3)</code> would swap the positions of the Custom Script and the Change Property actions on the <code>onMouseOver</code> event. Calling <code>dw.behaviorInspector.moveBehaviorUp()</code> for any other position would have no effect because the <code>onClick</code> and <code>onMouseOut</code> events each have only one behavior associated with them, and the behavior at position 2 is already at the top of the <code>onMouseOver</code> group.



## dreamweaver.behaviorInspector.setSelectedBehavior()

**Availability** 3.0

**Description** Selects the action at the specified position in the Behavior inspector.

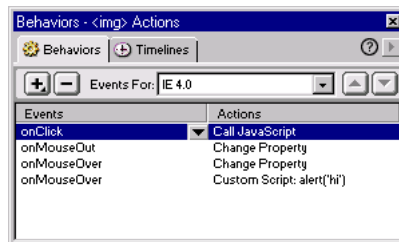
**Arguments** *positionIndex*

The argument is the position of the action in the Behavior inspector. The first action in the list is at position 0. To deselect all actions, specify a *positionIndex* of -1. Specifying a position for which no action exists is equivalent to specifying -1.

**Returns** Nothing.

**Enabler** None.

**Example** Assuming the Behavior inspector setup shown in this example, calling `dw.behaviorInspector.setSelection(3)` would select the Change Property action associated with the onMouseOut event.





## Clipboard functions

Clipboard functions are all related to cutting, copying, and pasting. On the Macintosh, some clipboard functions can also apply to edit fields in dialog boxes and floating palettes. Functions that can operate in edit fields are implemented both as methods of the `dreamweaver` object and as methods of the `dom` object. The `dreamweaver` version of the function operates on the selection in the window that has focus: the current Document window, the HTML inspector, or the Site window. On the Macintosh, the function can also operate on the selection in an edit field if the field has focus. The `dom` version of the function always operates on the selection in the specified document.

### `dom.clipCopy()`

<b>Availability</b>	3.0
<b>Description</b>	Copies the selection, including any HTML markup that defines the selection, to the clipboard.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dom.clipCopyText()`

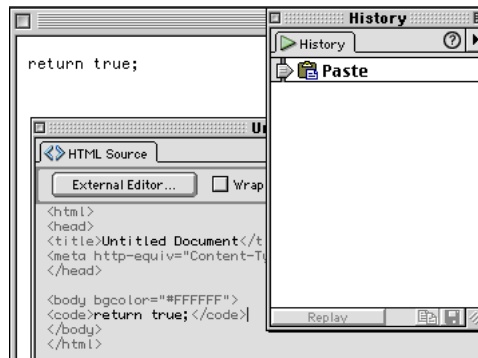
<b>Availability</b>	3.0
<b>Description</b>	Copies the selected text to the clipboard, ignoring any HTML markup.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dom.canClipCopyText()</code>

### `dom.clipCut()`

<b>Availability</b>	3.0
<b>Description</b>	Removes the selection, including any HTML markup that defines the selection, to the clipboard.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

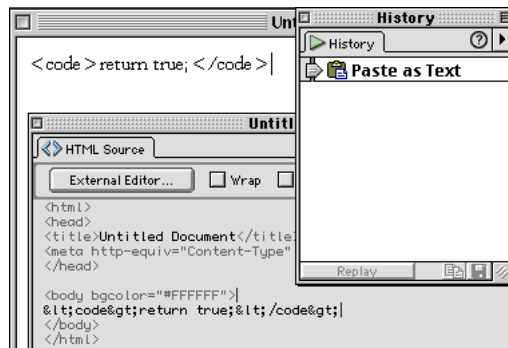
## dom.clipPaste()

<b>Availability</b>	3.0
<b>Description</b>	Pastes the contents of the clipboard into the current document at the current insertion point or in place of the current selection. If the clipboard contains HTML, it is interpreted as such.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canClipPaste()
<b>Example</b>	If the clipboard contains <code>&lt;code&gt;return true;&lt;/code&gt;</code> , a call to <code>dw.getDocumentDOM().clipPaste()</code> would result in the following:



## dom.clipPasteText()

<b>Availability</b>	3.0
<b>Description</b>	Pastes the contents of the clipboard into the current document at the current insertion point or in place of the current selection, replacing any linefeeds in the clipboard content with BR tags. If the clipboard contains HTML, it is not interpreted; angle brackets are pasted as &lt; and &gt;.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canClipPasteText()
<b>Example</b>	If the clipboard contains <code>&lt;code&gt;return true;&lt;/code&gt;</code> , a call to <code>dw.getDocumentDOM().clipPasteText()</code> would result in the following:



## dreamweaver.clipCopy()

<b>Availability</b>	3.0
<b>Description</b>	Copies the current selection (from whichever Document window, dialog box, floating palette, or Site window pane has focus) to the clipboard.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.canClipCopy()

### **dreamweaver.clipCut()**

<b>Availability</b>	3.0
<b>Description</b>	Removes the selection (from whichever Document window, dialog box, floating palette, or Site window pane has focus) to the clipboard.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.canClipCut()

### **dreamweaver.clipPaste()**

<b>Availability</b>	3.0
<b>Description</b>	Pastes the contents of the clipboard into the current document, dialog box, floating palette, or Site window pane.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.canClipPaste()

### **dreamweaver.getClipboardText()**

<b>Availability</b>	3.0
<b>Description</b>	Gets all the text that is stored on the clipboard.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the contents of the clipboard, if the clipboard contains text (which may be HTML); otherwise, nothing.
<b>Enabler</b>	None.

## Command functions

Command functions help you make the most of the files in the Configuration/Commands folder. They manage the Command menu and call commands from other types of extension files.

### `dreamweaver.editCommandList()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the Edit Command List dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.runCommand()`

<b>Availability</b>	3.0
<b>Description</b>	Executes the specified command. To the user, the effect is the same as choosing the command from a menu; if a dialog box is associated with the command, it appears (and the command script blocks other edits until the user dismisses the dialog box). This function provides the ability to call a command from another extension file.

**Note:** This function can be called only within the `objectTag()` function or in any script in a command, property inspector file, or menu.

<b>Arguments</b>	<i>commandFile</i> , { <i>commandArg1</i> }, { <i>commandArg2</i> },...{ <i>commandArgN</i> } <ul style="list-style-type: none"><li>• The argument is the name of a file in the Configuration/Commands folder.</li><li>• The second and remaining arguments are passed to <i>commandFile</i> as arguments.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	<p>You can write a custom property inspector for tables that lets users get to the Format Table command from a button on the inspector by calling the following function from the button's <code>onClick</code> event handler:</p> <pre>function callFormatTable(){     dw.runCommand('Format Table.htm'); }</pre>

## Conversion functions

Conversion functions deal with converting tables to layers, layers to tables, and Cascading Style Sheets (CSS) styles to HTML markup. Each function exactly duplicates the behavior of one of the conversion commands in the File or Modify menu.

### **dom.convertLayersToTable()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Convert Layers to Table dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canConvertLayersToTable()

### **dom.convertTablesToLayers()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Convert Tables to Layers dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canConvertTablesToLayers()

### **dom.convertTo30()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Convert to 3.0 Compatible dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## CSS style functions

CSS style functions handle the application, removal, creation, and deletion of CSS styles. Methods of the `dreamweaver.cssStylePalette` object either control or act on the selection in the Style palette, not in the current document.

### `dom.applyCSSStyle()`

<b>Availability</b>	3.0
<b>Description</b>	Applies the specified style to the specified element. This function is valid only for the active document.
<b>Arguments</b>	<p><i>elementNode</i>, <i>styleName</i>, {<i>classOrID</i>}</p> <ul style="list-style-type: none"><li>• The first argument is an element node in the DOM. If <i>elementNode</i> is specified as NULL or empty string ("), the function acts on the current selection.</li><li>• The second argument is the name of a CSS style.</li><li>• The third argument is the attribute with which the style should be applied (either "class" or "id"). If <i>elementNode</i> is specified as NULL or empty string and no tag exactly surrounds the selection, the style is applied using SPAN tags. If the selection is an insertion point, Dreamweaver uses heuristics to determine to which tag the style should be applied.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	<p>The following code applies the red style to the selection, either by surrounding the selection with SPAN tags or by applying a CLASS attribute to the tag that surrounds the selection:</p> <pre>var theDOM = dreamweaver.getDocumentDOM('document'); theDOM.applyCSSStyle("", 'red');</pre>

### dom.removeCSSStyle()

<b>Availability</b>	3.0
<b>Description</b>	Removes the CLASS or ID attribute from the specified element, or removes the SPAN tag that completely surrounds the specified element. This function is valid only for the active document.
<b>Arguments</b>	<i>elementNode</i> , { <i>classOrID</i> }
	<ul style="list-style-type: none"><li>• The first argument is an element node in the DOM. If <i>elementNode</i> is specified as an empty string (""), the function acts on the current selection.</li><li>• The second argument is the attribute that should be removed (either "class" or "id"). If <i>classOrID</i> is not specified, it defaults to "class". If no CLASS attribute is defined for <i>elementNode</i>, then the SPAN tag surrounding <i>elementNode</i> is removed.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### dreamweaver.cssStylePalette.deleteSelectedStyle()

<b>Availability</b>	3.0
<b>Description</b>	Deletes from the document the style that is currently selected in the Style palette.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### dreamweaver.cssStylePalette.duplicateSelectedStyle()

<b>Availability</b>	3.0
<b>Description</b>	Duplicates the style that is currently selected in the Style palette and displays the Duplicate Style dialog box to let the user assign a name or selector to the new style.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.



### `dreamweaver.cssStylePalette.editSelectedStyle()`

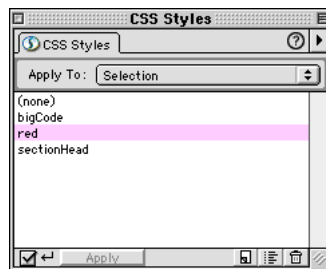
<b>Availability</b>	3.0
<b>Description</b>	Opens the Style Definition dialog box for the style that is currently selected in the Style palette.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.cssStylePalette.editStyleSheet()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the Edit Style Sheet dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

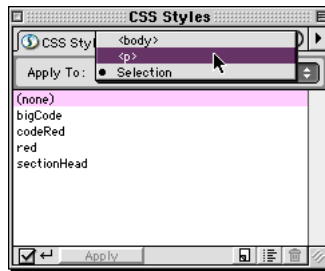
### `dreamweaver.cssStylePalette.getSelectedStyle()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the name of the style that is currently selected in the Style palette.
<b>Arguments</b>	None.
<b>Returns</b>	A string representing the name of the style, or an empty string if no style is selected.
<b>Enabler</b>	None.
<b>Example</b>	If the style red is selected, as shown, a call to <code>dw.cssStylePalette.getSelectedStyle()</code> returns "red".



## dreamweaver.cssStylePalette.getSelectedTarget()

<b>Availability</b>	3.0
<b>Description</b>	Gets the selected element in the Apply To pop-up menu at the top of the Style palette.
<b>Arguments</b>	None.
<b>Returns</b>	The object to which the style should be applied, or NULL if the target is the current selection.
<b>Enabler</b>	None.
<b>Example</b>	Before applying a style, use <code>dw.cssStylePalette.getSelectedTarget()</code> in case the user has changed the target, as shown.

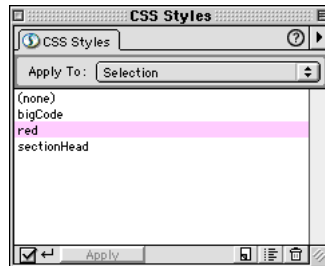


For example:

```
var currDOM = dw.getDocumentDOM();  
currDOM.applyCSSStyle(dw.cssStylePalette.getSelectedTarget(), "codeRed");
```

## `dreamweaver.cssStylePalette.getStyles()`

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of all the class styles in the active document.
<b>Arguments</b>	None.
<b>Returns</b>	An array of strings representing the names of all class styles in the document.
<b>Enabler</b>	None.
<b>Example</b>	<p>Assuming the CSS Style palette setup shown in this example, a call to <code>dw.cssStylePalette.getStyles()</code> would return an array containing the following strings:</p> <ul style="list-style-type: none"><li>• "bigCode"</li><li>• "red"</li><li>• "sectionHead"</li></ul>



## `dreamweaver.cssStylePalette.newStyle()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the New Style dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## External application functions

External application functions handle operations related to the browsers and external editors defined in the Preview in Browser and External Editors preferences. These functions let you get information about these external applications and open files with them.

### **`dreamweaver.browseDocument()`**

**Availability** 2.0, enhanced in 3.0

**Description** Opens the specified URL in the specified browser.

**Arguments** *fileName*, {*browserName*}

- The first argument is the name of the file to be opened, expressed as an absolute URL.
- The second argument, added in Dreamweaver 3, is the name of a browser as defined in the Preview in Browser preferences. If omitted, this argument defaults to the user's primary browser.

**Returns** Nothing.

**Enabler** None.

**Example** The following function uses `dreamweaver.browseDocument()` to open the Hotwired home page in a browser:

```
function goToHotwired(){
    dreamweaver.browseDocument('http://www.hotwired.com/');
}
```

In Dreamweaver 3, you can expand this operation to open the document in Internet Explorer using the following code:

```
function goToHotwired(){
    var prevBrowsers = dw.getBrowserList();
    var theBrowser = "";
    for (var i=1; i < prevBrowsers.length; i+2){
        if (prevBrowsers[i].indexOf('Iexplore.exe') != -1){
            theBrowser = prevBrowsers[i];
            break;
        }
    }
    dw.browseDocument('http://www.hotwired.com/',theBrowser);
}
```

For more information on `dw.getBrowserList()`, see “`dreamweaver.getBrowserList()`” on page 45.

### **dreamweaver.getBrowserList()**

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of all the browsers in the Preview in Browser submenu.
<b>Arguments</b>	None.
<b>Returns</b>	An array containing a pair of strings for each browser in the list. The first string in each pair is the name of the browser, and the second string is its location on the user's machine, expressed as a file:// URL. If no browsers appear in the submenu, the function returns nothing.
<b>Enabler</b>	None.

### **dreamweaver.getExtensionEditorList()**

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of editors for the specified file from the External Editors preferences.
<b>Arguments</b>	<i>fileURL</i>  The argument can be a complete file:// URL, a file name, or a file extension (including the period).
<b>Returns</b>	An array containing a pair of strings for each editor in the list. The first string in each pair is the name of the editor, and the second string is its location on the user's machine, expressed as a file:// URL. If no editors appear in the preferences, the function returns an array of one empty string.
<b>Enabler</b>	None.
<b>Example</b>	A call to <code>dw.getExtensionEditorList(".gif")</code> might return an array containing the following strings: <ul style="list-style-type: none"><li>• "Fireworks 3"</li><li>• "file:///C:/Program Files/Macromedia/Fireworks 3/Fireworks 3.exe"</li></ul>

### **dreamweaver.getPrimaryBrowser()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the path to the primary browser.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the path on the user's hard drive to the primary browser, expressed as a file:// URL, or nothing if no primary browser is defined.
<b>Enabler</b>	None.

### **`dreamweaver.getPrimaryExtensionEditor()`**

<b>Availability</b>	3.0
<b>Description</b>	Gets the primary editor for the specified file.
<b>Arguments</b>	<i>fileURL</i>
<b>Returns</b>	An array containing a pair of strings. The first string in the pair is the name of the editor, and the second string is its location on the user's machine, expressed as a file:// URL. If no primary editor is defined, the function returns an array of one empty string.
<b>Enabler</b>	None.

### **`dreamweaver.getSecondaryBrowser()`**

<b>Availability</b>	3.0
<b>Description</b>	Gets the path to the secondary browser.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the path on the user's hard drive to the secondary browser, expressed as a file:// URL, or nothing if no secondary browser is defined.
<b>Enabler</b>	None.

### **`dreamweaver.openWithApp()`**

<b>Availability</b>	3.0
<b>Description</b>	Opens the specified file with the specified application.
<b>Arguments</b>	<i>fileURL, appURL</i> <ul style="list-style-type: none"><li>• The first argument is the path to the file to be opened, expressed as a file:// URL.</li><li>• The second argument is the path to the application that is to open the file, expressed as a file:// URL.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.openWithBrowseDialog()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Select External Editor dialog box to let the user choose the application with which to open the specified file.
<b>Arguments</b>	<i>fileURL</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.openWithExternalTextEditor()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the current document in the external text editor specified in the External Editors preferences.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.openWithImageEditor()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the specified file with the specified image editor.  <b>Note:</b> This function invokes a special Fireworks integration mechanism that returns information to the active document if Fireworks is specified as the image editor. To prevent errors if no document is active, this function should never be called from the Site window.
<b>Arguments</b>	<i>fileURL, appURL</i> <ul style="list-style-type: none"><li>• The first argument is the path to the file to be opened, expressed as a file:// URL.</li><li>• The second argument is the path to the application with which to open the file, expressed as a file:// URL.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## File manipulation functions

File manipulation functions deal with creating, opening, and saving documents, and with exporting cascading style sheets to external files. They accomplish such tasks as browsing for files or folders, creating files based on templates, closing documents, and getting information about recently opened files.

### **dreamweaver.browseForFileURL()**

<b>Availability</b>	1.0, enhanced in 2.0 and 3.0
<b>Description</b>	Opens the specified type of dialog box with the specified label in the title bar.
<b>Arguments</b>	<i>openSelectOrSave</i> , <i>{titleBarLabel}</i> , <i>{bShowPreviewPane}</i> , <i>{bSupressSiteRootWarnings}</i> <ul style="list-style-type: none"><li>• The first argument indicates the type of dialog box: open, select, or save.</li><li>• The second argument, added in Dreamweaver 2, is the label that should appear in the title bar of the dialog box. If this argument is omitted, Dreamweaver uses the default label supplied by the operating system.</li><li>• The third argument, added in Dreamweaver 2, is a Boolean value indicating whether to display the image preview pane in the dialog box. If this argument is TRUE, the dialog box filters for image files; if omitted, it defaults to FALSE.</li><li>• The fourth argument, added in Dreamweaver 3, is a Boolean value indicating whether to suppress warnings about the selected file being outside the site root. If this argument is omitted, it defaults to FALSE.</li></ul>
<b>Returns</b>	A string containing the name of the file, expressed as a file:// URL.
<b>Enabler</b>	None.

### **dreamweaver.browseForFolderURL()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Choose Folder dialog box with the specified label in the title bar.
<b>Arguments</b>	<i>{titleBarLabel}</i> , <i>{directoryToStartIn}</i> <ul style="list-style-type: none"><li>• The first argument is the label that should appear in the title bar of the dialog box. If omitted, <i>titleBarLabel</i> defaults to “Choose Folder.”</li><li>• The second argument is the path to the directory to start in, expressed as a file:// URL.</li></ul>
<b>Returns</b>	A string containing the name of the folder, expressed as a file:// URL.
<b>Enabler</b>	None.
<b>Example</b>	The following code returns the URL of a folder: <pre>return dw.browseForFolderURL('Select a Folder', dw.getSiteRoot());</pre>



## **dreamweaver.closeDocument()**

<b>Availability</b>	3.0
<b>Description</b>	Closes the specified document.
<b>Arguments</b>	<p><i>documentObject</i></p> <p>The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code>). If <i>documentObject</i> refers to the active document, the Document window might not close until the script that calls this function finishes executing.</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## **dreamweaver.createDocument()**

<b>Availability</b>	2.0
<b>Description</b>	<p>Opens a new document either in the same window or in a new window. The new document becomes the active document.</p> <p><b>Note:</b> This function can be called only from <code>menus.xml</code> or a command or property inspector file. If a behavior action or object tries to call this function, Dreamweaver displays an error message.</p>
<b>Arguments</b>	<p><i>{bOpenInSameWindow}</i></p> <p>The argument is a Boolean value indicating whether to open the new document in the current window. If <i>bOpenInSameWindow</i> is <code>FALSE</code> or omitted, or the function is called on the Macintosh, the new document opens in a separate window.</p>
<b>Returns</b>	The document object for the newly created document. This is the same value returned by <code>dreamweaver.getDocumentDOM()</code> .
<b>Enabler</b>	None.

## **dreamweaver.exportCSS()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Export Styles as CSS File dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canExportCSS()</code>

### **dreamweaver.exportEditableRegionsAsXML()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Export Editable Regions as XML dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.getRecentFileList()**

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of all the files in the recent files list at the bottom of the File menu.
<b>Arguments</b>	None.
<b>Returns</b>	An array of strings representing the paths of the most recently accessed files, expressed as file:// URLs. If there are no recent files, the function returns nothing.
<b>Enabler</b>	None.

### **dreamweaver.importXMLIntoTemplate()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Import XML dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.newFromTemplate()**

<b>Availability</b>	3.0
<b>Description</b>	Creates a new document from the specified template. If no argument is supplied, the Select Template dialog box appears.
<b>Arguments</b>	<i>{templateURL}</i>  The argument is the path to a template in the current site, expressed as a file:// URL.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## **dreamweaver.openDocument()**

<b>Availability</b>	2.0
<b>Description</b>	Opens a document for editing in a new Dreamweaver window and gives it the focus. To the user, the effect is the same as choosing File > Open and selecting a file. If the specified file is already open, the window containing the document comes to the front. The window containing the specified file becomes the currently selected document. In Dreamweaver 2, if check in/check out is enabled, the file is checked out before it is opened. In Dreamweaver 3, you must use <code>dreamweaver.openDocumentFromSite()</code> to get this behavior.

**Note:** This function cannot be called from behavior action or object files. An error will result.

<b>Arguments</b>	<i>fileName</i>  The argument is the name of the file to be opened, expressed as a URL. If the URL is relative, it is relative to the file containing the script that called this function.
<b>Returns</b>	The document object for the specified file. This is the same value that is returned by <code>dreamweaver.getDocumentDOM()</code> .
<b>Enabler</b>	None.

## **dreamweaver.openDocumentFromSite()**

<b>Availability</b>	3.0
<b>Description</b>	Opens a document for editing in a new Dreamweaver window and gives it the focus. To the user, the effect is the same as double-clicking a file in the Site window. If the specified file is already open, the window containing the document comes to the front. The window containing the specified file becomes the currently selected document.

**Note:** This function cannot be called from behavior action or object files. An error will result.

<b>Arguments</b>	<i>fileName</i>  The argument is the name of the file to be opened, expressed as a URL. If the URL is relative, it is relative to the file containing the script that called this function.
<b>Returns</b>	The document object for the specified file. This is the same value that is returned by <code>dreamweaver.getDocumentDOM()</code> .
<b>Enabler</b>	None.

### **dreamweaver.openInFrame()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Open In Frame dialog box. When the user selects a document, it is opened into the active frame.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canOpenInFrame()</code>

### **dreamweaver.releaseDocument()**

<b>Availability</b>	2.0
<b>Description</b>	<p>Explicitly releases a previously referenced document from memory.</p> <p>Documents referenced by <code>dreamweaver.getObjectTags()</code>, <code>dreamweaver.getObjectRefs()</code>, <code>dreamweaver.getDocumentPath()</code>, or <code>dreamweaver.getDocumentDOM()</code> are automatically released when the script that contains the call finishes executing. If the script opens many documents, you must use this function to explicitly release documents before finishing the script to avoid running out of memory.</p> <p><b>Note:</b> This function is relevant only for documents that were referenced by a URL, that are not currently open in a frame or Document window, and that are not extension files. (Extension files are loaded into memory at startup and are not released until you quit Dreamweaver.)</p>
<b>Arguments</b>	<p><i>documentObject</i></p> <p>The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code>).</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.revertDocument()**

<b>Availability</b>	3.0
<b>Description</b>	Reverts the specified document to the previously saved version.
<b>Arguments</b>	<p><i>documentObject</i></p> <p>The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code>).</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canRevertDocument()</code>

## **dreamweaver.saveAll()**

<b>Availability</b>	3.0
<b>Description</b>	Saves all open documents, opening the Save As dialog box for any documents that have not previously been saved.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.canSaveAll()

## **dreamweaver.saveDocument()**

<b>Availability</b>	2.0
<b>Description</b>	Saves the specified file on a local drive.

**Note:** In Dreamweaver 2, if the file is read-only, Dreamweaver tries to check it out. If the document is still read-only after this attempt—or if it cannot be created—an error message appears.

<b>Arguments</b>	<i>documentObject</i> , { <i>fileURL</i> }
	<ul style="list-style-type: none"><li>• The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code>).</li><li>• The second argument is a URL representing a location on a local drive. If the URL is relative, it is relative to the extension file. In Dreamweaver 2, this argument is required. If <i>fileURL</i> is omitted in Dreamweaver 3, the file is saved to its current location if it has been previously saved; otherwise, a Save dialog box appears.</li></ul>
<b>Returns</b>	A Boolean value indicating success (TRUE) or failure (FALSE).
<b>Enabler</b>	dreamweaver.canSaveDocument()

## **dreamweaver.saveDocumentAs()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Save As dialog box.
<b>Arguments</b>	<i>documentObject</i>  The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code> ).
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.saveDocumentAsTemplate()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Save As Template dialog box.
<b>Arguments</b>	<i>documentObject</i>  The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code> ).
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canSaveDocumentAsTemplate()</code>

### **dreamweaver.saveFrameset()**

<b>Availability</b>	3.0
<b>Description</b>	Saves the specified frameset, or opens the Save As dialog box if the frameset has not previously been saved.
<b>Arguments</b>	<i>documentObject</i>  The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code> ).
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canSaveFrameset()</code>

### **dreamweaver.saveFramesetAs()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Save As dialog box for the frameset file that includes the specified DOM.
<b>Arguments</b>	<i>documentObject</i>  The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code> ).
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canSaveFramesetAs()</code>

## Find/replace functions

Find/replace functions handle find and replace operations. They cover both basic functionality, such as finding the next instance of a search pattern, and complex replacement operations that require no user interaction.

### `dreamweaver.findNext()`

<b>Availability</b>	3.0
<b>Description</b>	Finds the next instance of the search string that was specified previously by <code>dreamweaver.setUpFind()</code> , by <code>dreamweaver.setUpComplexFind()</code> , or by the user in the Find dialog box, and selects the instance in the document.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canFindNext()</code>

### `dreamweaver.replace()`

<b>Availability</b>	3.0
<b>Description</b>	Verifies that the current selection matches the search criteria that was specified previously by <code>dreamweaver.setUpFindReplace()</code> , by <code>dreamweaver.setUpComplexFindReplace()</code> , or by the user in the Replace dialog box; then replaces it with the content specified in that query.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.replaceAll()`

<b>Availability</b>	3.0
<b>Description</b>	Replaces each section of the current document that matches the search criteria that was specified previously by <code>dreamweaver.setUpFindReplace()</code> , by <code>dreamweaver.setUpComplexFindReplace()</code> , or by the user in the Replace dialog box, with the content specified in that query.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## **dreamweaver.setUpComplexFind()**

<b>Availability</b>	3.0
<b>Description</b>	Prepares for an advanced text or tag search by loading the specified XML query.
<b>Arguments</b>	<p><i>xmlQueryString</i></p> <p>The argument is a string of XML code beginning with &lt;dwquery&gt; and ending with &lt;/dwquery&gt;. (To get a string of the proper format, set up the query in the Find dialog box, click the Save Query button, open the query file in a text editor, and copy everything from the beginning of the &lt;dwquery&gt; tag to the end of the &lt;/dwquery&gt; tag.)</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	<p>The first line of the following code sets up a tag search and specifies that the scope of the search should be the current document; the second line performs the search operation:</p> <pre>dw.setUpComplexFind('&lt;dwquery&gt; &lt;queryparams matchcase="false" ignorewhitespace="true" useregexp="false"/&gt; &lt;find&gt; &lt;qtag qname="a"&gt; &lt;qattribute qname="href" qcompare="=" qvalue="#"&gt; &lt;/qattribute&gt; &lt;qattribute qname="onMouseOut" qcompare="=" qvalue="" qnegate="true"&gt; &lt;/qattribute&gt; &lt;/qtag&gt; &lt;/find&gt; &lt;/dwquery&gt;'); dw.findNext();</pre>



## **dreamweaver.setUpComplexFindReplace()**

<b>Availability</b>	3.0
<b>Description</b>	Prepares for an advanced text or tag search by loading the specified XML query.
<b>Arguments</b>	<p><i>xmlQueryString</i></p> <p>The argument is a string of XML code beginning with &lt;dwquery&gt; and ending with &lt;/dwquery&gt;. (To get a string of the proper format, set up the query in the Find dialog box, click the Save Query button, open the query file in a text editor, and copy everything from the beginning of the &lt;dwquery&gt; tag to the end of the &lt;/dwquery&gt; tag.)</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	<p>The first statement in the following code sets up a tag search and specifies that the scope of the search should be four files; the second statement performs the search and replace operation:</p> <pre>dw.setUpComplexFindReplace('&lt;dwquery&gt;&lt;queryparams matchcase="false" ignorewhitespace="true" useregexp="false"/&gt;&lt;find&gt;&lt;qtag qname="a"&gt;&lt;qattribute qname="href" qcompare="=" qvalue="#"&gt;&lt;/qattribute&gt;&lt;qattribute qname="onMouseOut" qcompare="=" qvalue="" qnegate="true"&gt;&lt;/qattribute&gt;&lt;/qtag&gt;&lt;/find&gt;&lt;replace action="setAttribute" param1="onMouseOut" param2="this.style.color='#000000';this.style.fontWeight='normal'"/&gt; &lt;/dwquery&gt;'); dw.replaceAll();</pre>

## **dreamweaver.setUpFind()**

<b>Availability</b>	3.0
<b>Description</b>	Prepares for a text or HTML source search by defining the search parameters for a subsequent <code>dw.findNext()</code> operation.
<b>Arguments</b>	<p><i>searchObject</i></p> <p>The argument is an object for which the following properties can be defined:</p> <ul style="list-style-type: none"><li>• <i>searchString</i> is the text to search for.</li><li>• <i>searchSource</i> is a Boolean value indicating whether to search the HTML source.</li><li>• <i>{matchCase}</i> is a Boolean value indicating whether the search is case-sensitive. If this property is not explicitly set, it defaults to <code>FALSE</code>.</li><li>• <i>{ignoreWhitespace}</i> is a Boolean value indicating whether white space differences should be ignored. <i>ignoreWhitespace</i> defaults to <code>FALSE</code> if <i>useRegularExpressions</i> is <code>TRUE</code> and <code>TRUE</code> if <i>useRegularExpressions</i> is <code>FALSE</code>.</li><li>• <i>{useRegularExpressions}</i> is a Boolean value indicating whether the <i>searchString</i> uses regular expressions. If this property is not explicitly set, it defaults to <code>FALSE</code>.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	<p>The following code demonstrates three ways to create a <i>searchObject</i> object:</p> <pre>var searchParams; searchParams.searchString = 'bgcolor="#FFCCFF"; searchParams.searchSource = true; dw.setUpFind(searchParams);  var searchParams = {searchString: 'bgcolor="#FFCCFF"', searchSource: true}; dw.setUpFind(searchParams);  dw.setUpFind({searchString: 'bgcolor="#FFCCFF"', searchSource: true});</pre>

## **dreamweaver.setUpFindReplace()**

<b>Availability</b>	3.0
<b>Description</b>	Prepares for a text or HTML source search by defining the search parameters and the scope for a subsequent <code>dw.replace()</code> or <code>dw.replaceAll()</code> operation.
<b>Arguments</b>	<p><i>searchObject</i></p> <p>The argument is an object for which the following properties can be defined:</p> <ul style="list-style-type: none"><li>• <i>searchString</i> is the text to search for.</li><li>• <i>replaceString</i> is the text with which to replace the selection.</li><li>• <i>searchSource</i> is a Boolean value indicating whether to search the HTML source.</li><li>• <i>{matchCase}</i> is a Boolean value indicating whether the search is case-sensitive. If this property is not explicitly set, it defaults to <code>FALSE</code>.</li><li>• <i>{ignoreWhitespace}</i> is a Boolean value indicating whether white space differences should be ignored. <i>ignoreWhitespace</i> defaults to <code>FALSE</code> if <i>useRegularExpressions</i> is <code>TRUE</code> and <code>TRUE</code> if <i>useRegularExpressions</i> is <code>FALSE</code>.</li><li>• <i>{useRegularExpressions}</i> is a Boolean value indicating whether the <i>searchString</i> uses regular expressions. If this property is not explicitly set, it defaults to <code>FALSE</code>.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	<p>The following code demonstrates three ways to create a <i>searchObject</i> object:</p> <pre>var searchParams; searchParams.searchString = 'bgcolor="#FFCCFF"; searchParams.replaceString = 'bgcolor="#CCFFCC"; searchParams.searchSource = true; dw.setUpFindReplace(searchParams);</pre>

```
var searchParams = {searchString: 'bgcolor="#FFCCFF", replaceString: 'bgcolor="#CCFFCC",  
searchSource: true};  
dw.setUpFindReplace(searchParams);
```

```
dw.setUpFindReplace({searchString: 'bgcolor="#FFCCFF", replaceString:  
'bgcolor="#CCFFCC", searchSource: true});
```

## **dreamweaver.showFindDialog()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Find dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canShowFindDialog()</code>

## `dreamweaver.showFindReplaceDialog()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the Replace dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canShowFindDialog()</code>

## Frame and frameset functions

Frame and frameset functions cover only two tasks: getting the names of the frames in a frameset, and splitting a frame in two.

### `dom.getFrameNames()`

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of all the named frames in the frameset.
<b>Arguments</b>	None.
<b>Returns</b>	An array of strings, each a name of a frame in the current frameset. Any unnamed frames are skipped. If none of the frames in the frameset is named, an empty array is returned.
<b>Enabler</b>	None.
<b>Example</b>	<p>For a document containing four frames, two of which are named, a call to <code>dw.getDocumentDOM().getFrameNames()</code> might return an array containing the following strings:</p> <ul style="list-style-type: none"><li>• "navframe"</li><li>• "main_content"</li></ul>

### `dom.splitFrame()`

<b>Availability</b>	3.0
<b>Description</b>	Splits the selected frame vertically or horizontally.
<b>Arguments</b>	<p><i>splitDirection</i></p> <p>The argument must be one of the following: "up", "down", "left", or "right".</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dom.canSplitFrame()</code>

## General editing functions

General editing functions handle common editing tasks that happen in the Document window. These functions insert text, HTML, and objects; apply, change, and remove font and character markup; modify tags and attributes; and more.

### `dom.applyCharacterMarkup()`

<b>Availability</b>	3.0
<b>Description</b>	Applies the specified type of character markup to the selection. If the selection is an insertion point, applies the specified character markup to any subsequently-typed text.
<b>Arguments</b>	<p><i>tagName</i></p> <p>The argument is the tag name associated with the character markup. It must be one of the following strings: "b", "cite", "code", "dfn", "em", "i", "kbd", "samp", "s", "strong", "tt", "u", or "var".</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dom.applyFontMarkup()`

<b>Availability</b>	3.0
<b>Description</b>	Applies the FONT tag and the specified attribute and value to the current selection.
<b>Arguments</b>	<p><i>attribute, value</i></p> <ul style="list-style-type: none"><li>• The first argument must be "face", "size", or "color".</li><li>• The second argument is the value that should be assigned to the attribute; for example, "Arial, Helvetica, sans-serif", "5", or "#FF0000".</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dom.deleteSelection()`

<b>Availability</b>	3.0
<b>Description</b>	Deletes the selection in the document.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.editAttribute()**

<b>Availability</b>	3.0
<b>Description</b>	Displays the appropriate interface for editing the specified attribute. In most cases, this is a dialog box. This function is valid only for the active document.
<b>Arguments</b>	<i>attribute</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.exitBlock()**

<b>Availability</b>	3.0
<b>Description</b>	Exits the current paragraph or heading block, leaving the cursor outside of all block elements.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.getFontMarkup()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the value of the specified attribute of the FONT tag for the current selection.
<b>Arguments</b>	<i>attribute</i>  The argument must be "face", "size", or "color".
<b>Returns</b>	A string containing the value of the specified attribute, or an empty string if the attribute is not set.
<b>Enabler</b>	None.

### **dom.getLinkHref()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the link that surrounds the current selection. This function is equivalent to looping through the parents and grandparents of the current node until a link is found, and then calling <code>getAttribute("HREF")</code> on the link.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the name of the linked file, expressed as a file:// URL.
<b>Enabler</b>	None.

### **dom.getLinkTarget()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the target of the link that surrounds the current selection. This function is equivalent to looping through the parents and grandparents of the current node until a link is found, and then calling <code>getAttribute('TARGET')</code> on the link.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the value of the <code>TARGET</code> attribute for the link, or an empty string if no target is specified.
<b>Enabler</b>	None.

### **dom.getListTag()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the style of the selected list.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the tag associated with the list (" <code>ul</code> ", " <code>ol</code> ", or " <code>dl</code> "), or an empty string if no tag is associated with the list. This value is always returned in lowercase letters.
<b>Enabler</b>	None.

### **dom.getTextAlignment()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the alignment of the block that contains the selection.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the value of the <code>ALIGN</code> attribute for the tag associated with the block, or an empty string if the <code>ALIGN</code> attribute is not set for the tag. This value is always returned in lowercase letters.
<b>Enabler</b>	None.

### **dom.getTextFormat()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the block format of the selected text.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the block tag associated with the text (for example, " <code>p</code> ", " <code>h1</code> ", " <code>pre</code> ", and so on, or an empty string if no block tag is associated with the selection). This value is always returned in lowercase letters.
<b>Enabler</b>	None.

## dom.hasCharacterMarkup()

<b>Availability</b>	3.0
<b>Description</b>	Checks whether the selection already has the specified character markup.
<b>Arguments</b>	<p><i>markupTagName</i></p> <p>The argument is the name of the tag you're checking for. It must be one of the following strings: "b", "cite", "code", "dfn", "em", "i", "kbd", "samp", "s", "strong", "tt", "u", or "var".</p>
<b>Returns</b>	A Boolean value indicating whether the entire selection has the specified character markup. The function returns FALSE if only part of the selection has the specified markup.
<b>Enabler</b>	None.

## dom.indent()

<b>Availability</b>	3.0
<b>Description</b>	Indents the selection using BLOCKQUOTE tags. If the selection is inside a list, indents the selection by nesting another list of the same type inside the current list.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

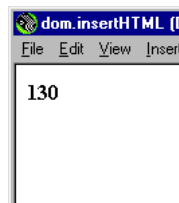


## dom.insertHTML()

<b>Availability</b>	3.0
<b>Description</b>	Inserts HTML content into the document at the current insertion point.
<b>Arguments</b>	<i>contentToInsert</i> , { <i>bReplaceCurrentSelection</i> }
	<ul style="list-style-type: none"><li>• The first argument is the content you want to insert.</li><li>• The second argument is a Boolean value indicating whether the content should replace the current selection. If <i>bReplaceCurrentSelection</i> is <i>False</i>, the content is inserted after the current selection.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	The following code inserts <code>&lt;b&gt;130&lt;/b&gt;</code> into the current document:

```
var theDOM = dw.getDocumentDOM();  
theDOM.insertHTML('<b>130</b>');
```

This appears in the Document window as:



## dom.insertObject()

<b>Availability</b>	3.0
<b>Description</b>	Inserts the specified object, prompting the user for parameters if necessary.
<b>Arguments</b>	<i>objectName</i>
	The argument is the name of an object in the Configuration/Objects folder.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	A call to <code>dreamweaver.getDocumentDOM().insertObject('Button');</code> inserts a form button into the current document at the current insertion point or after the current selection.

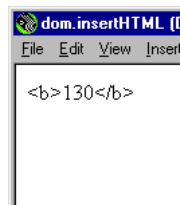
**Note:** Although object files can be stored in separate folders, it's important that their file names be unique. If a file called `Button.htm` exists in the Forms folder and also in the MyObjects folder, Dreamweaver cannot distinguish between them.

## dom.insertText()

<b>Availability</b>	3.0
<b>Description</b>	Inserts text content into the document at the current insertion point.
<b>Arguments</b>	<i>contentToInsert</i> , { <i>bReplaceCurrentSelection</i> }
	<ul style="list-style-type: none"><li>• The first argument is the content you want to insert.</li><li>• The second argument is a Boolean value indicating whether the content should replace the current selection. If <i>bReplaceCurrentSelection</i> is <i>False</i>, the content is inserted after the current selection.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	The following code inserts <code>&lt;b&gt;130&lt;/b&gt;</code> into the current document:

```
var theDOM = dw.getDocumentDOM();  
theDOM.insertText('<b>130</b>');
```

This appears in the Document window as:



## dom.newBlock()

<b>Availability</b>	3.0
<b>Description</b>	Creates a new block with the same tag and attributes as the block that contains the current selection, or creates a new paragraph if the cursor is outside of all blocks.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	If the current selection is inside a center-aligned paragraph, a call to <code>dreamweaver.getDocumentDOM().newBlock()</code> inserts <code>&lt;p align="center"&gt;</code> after the current paragraph.

### **dom.outdent()**

<b>Availability</b>	3.0
<b>Description</b>	Outdents the selection.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.removeCharacterMarkup()**

<b>Availability</b>	3.0
<b>Description</b>	Removes the specified type of character markup from the selection.
<b>Arguments</b>	<i>tagName</i>  The argument is the tag name associated with the character markup. It must be one of the following strings: "b", "cite", "code", "dfn", "em", "i", "kbd", "samp", "s", "strong", "tt", "u", or "var".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.removeFontMarkup()**

<b>Availability</b>	3.0
<b>Description</b>	Removes the specified attribute and its value from a FONT tag. If removing the attribute would leave only <FONT>, then the FONT tag is also removed.
<b>Arguments</b>	<i>attribute</i>  The argument must be "face", "size", or "color".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.removeLink()**

<b>Availability</b>	3.0
<b>Description</b>	Removes the hyperlink from the selection.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.resizeSelection()**

<b>Availability</b>	3.0
<b>Description</b>	Resizes the selected object to the specified dimensions. To resize a layer or hotspot, use <code>dom.resizeSelectionBy()</code> .
<b>Arguments</b>	<i>newWidth, newHeight</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setAttributeWithErrorChecking()**

<b>Availability</b>	3.0
<b>Description</b>	Sets the specified attribute to the specified value for the current selection, prompting the user if the value is of the wrong type, or if it is out of range. This function is valid only for the active document.
<b>Arguments</b>	<i>attribute, value</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setLinkHref()**

<b>Availability</b>	3.0
<b>Description</b>	Makes the selection a hyperlink, or changes the value of the link that surrounds the current selection.
<b>Arguments</b>	<i>linkHref</i>  The argument is the URL (document-relative path, root-relative path, or absolute URL) to link to. If the argument is omitted, the Select HTML File dialog box appears.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dom.canSetLinkHref()</code>

## dom.setLinkTarget()

<b>Availability</b>	3.0
<b>Description</b>	Sets the target of the link that surrounds the current selection. This function is equivalent to looping through the parents and grandparents of the current node until a link is found, and then calling <code>setAttribute('TARGET')</code> on the link.
<b>Arguments</b>	<i>{linkTarget}</i>  The argument is a string representing a frame or window name, or one of the reserved targets ("_self", "_parent", "_top", or "_blank"). If the argument is omitted, the Set Target dialog box appears.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.setListBoxKind()

<b>Availability</b>	3.0
<b>Description</b>	Changes the kind of the selected SELECT menu.
<b>Arguments</b>	<i>kind</i>  The argument must be either "menu" or "list box".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.showListPropertiesDialog()

<b>Availability</b>	3.0
<b>Description</b>	Opens the List Properties dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dom.canShowListPropertiesDialog()</code>

### **dom.setListTag()**

<b>Availability</b>	3.0
<b>Description</b>	Sets the style of the selected list.
<b>Arguments</b>	<i>listTag</i>  The argument is the tag associated with the list. It must be "ol", "ul", "dl", or an empty string.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setTextAlignment()**

<b>Availability</b>	3.0
<b>Description</b>	Sets the ALIGN attribute of the block that contains the selection to the specified value.
<b>Arguments</b>	<i>alignValue</i>  The argument must be "left", "center", or "right".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setTextFieldKind()**

<b>Availability</b>	3.0
<b>Description</b>	Sets the format of the selected text field.
<b>Arguments</b>	<i>fieldType</i>  The argument must be "input", "textarea", or "password".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.showFontColorDialog()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Color Picker dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.deleteSelection()**

<b>Availability</b>	3.0
<b>Description</b>	Deletes the selection in the active document or the Site window; or, on the Macintosh, the edit field that has focus in a dialog box or floating palette.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.canDeleteSelection()

### **dreamweaver.editFontList()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Edit Font List dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.getFontList()**

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of all the font groups that appear in the text Property inspector and in the Style Definition dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	An array of strings representing each item in the font list.
<b>Enabler</b>	None.
<b>Example</b>	<p>For the default installation of Dreamweaver, a call to <code>dw.getFontList()</code> would return an array containing the following items:</p> <ul style="list-style-type: none"><li>• "Arial, Helvetica, sans-serif"</li><li>• "Times New Roman, Times, serif"</li><li>• "Courier New, Courier, mono"</li><li>• "Georgia, Times New Roman, Times, serif"</li><li>• "Verdana, Arial, Helvetica, sans-serif"</li></ul>

## **dreamweaver.getKeyState()**

**Availability** 3.0

**Description** Determines whether the specified modifier key is down.

**Arguments** *key*

The argument must be one of the following: "Cmd", "Ctrl", "Alt", or "Shift". In Windows, "Cmd" and "Ctrl" both refer to the Control key; on the Macintosh, "Alt" refers to the Option key.

**Returns** A Boolean value indicating whether the key is down.

**Enabler** None.

**Example** The following code checks that both the Shift and Control keys (Windows) or Shift and Command keys (Macintosh) are down before performing an operation:

```
if (dw.getKeyState("Shift") && dw.getKeyState("Cmd")){  
    // execute code  
}
```



## Global application functions

Global application functions act on the entire application. They provide such functionality as quitting and accessing preferences.

### `dreamweaver.getShowDialogsOnInsert()`

<b>Availability</b>	3.0
<b>Description</b>	Checks whether the Show Dialog When Inserting Objects option is turned on in the General preferences.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the option is on.
<b>Enabler</b>	None.

### `dreamweaver.quitApplication()`

<b>Availability</b>	3.0
<b>Description</b>	Quits Dreamweaver after the script that calls this function finishes executing.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.showAboutBox()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the About box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## **dreamweaver.showPreferencesDialog()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Preferences dialog box.
<b>Arguments</b>	<i>{whichTab}</i>  The argument must be one of the following strings: "general", "external editors", "floaters", "fonts", "highlighting", "html colors", "html format", "html rewriting", "invisible elements", "layers", "browsers", "quick tag editor", "site ftp", "status bar", "css styles", and "translation". If Dreamweaver does not recognize the argument as a valid pane name, or if the argument is omitted, the dialog box opens to the last-active pane.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## **Global document functions**

Global document functions act on the entire document. They check spelling, check target browsers, set page properties, and determine correct object references for elements in the document.

### **dom.checkSpelling()**

<b>Availability</b>	3.0
<b>Description</b>	Checks the spelling in the document, opening the Check Spelling dialog box if necessary, and notifies the user when the check is complete.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.checkTargetBrowsers()**

<b>Availability</b>	3.0
<b>Description</b>	Runs a target browser check on the document. To run a target browser check on a folder or group of files, use <code>site.checkTargetBrowsers()</code> .
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.showPagePropertiesDialog()

<b>Availability</b>	3.0
<b>Description</b>	Opens the Page Properties dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dreamweaver.getElementRef()

<b>Availability</b>	2.0
<b>Description</b>	Gets the Netscape or IE object reference for a specific tag object in the DOM tree.
<b>Arguments</b>	<i>NSorIE</i> , <i>tagObject</i> <ul style="list-style-type: none"><li>• The first argument must be either "NS 4.0" or "IE 4.0". The DOM and rules for nested references differ in Navigator 4.0 and Internet Explorer 4.0. This argument specifies for which browser to return a valid reference.</li><li>• The second argument is a tag object in the DOM tree.</li></ul>
<b>Returns</b>	A string representing a valid JavaScript reference to the object, such as <code>document.layers['myLayer']</code> . <ul style="list-style-type: none"><li>• Dreamweaver returns correct references for Internet Explorer for A, AREA, APPLET, EMBED, DIV, SPAN, INPUT, SELECT, OPTION, TEXTAREA, OBJECT, and IMG tags.</li><li>• Dreamweaver returns correct references for Navigator for A, AREA, APPLET, EMBED, LAYER, ILAYER, SELECT, OPTION, TEXTAREA, OBJECT, and IMG tags, and for absolutely positioned DIV and SPAN tags. For DIV and SPAN tags that are not absolutely positioned, Dreamweaver returns "cannot reference &lt;tag&gt;".</li><li>• Dreamweaver does not return references for unnamed objects. If an object does not contain either a NAME or an ID attribute, then Dreamweaver returns "unnamed &lt;tag&gt;". If the browser does not support a reference by name, Dreamweaver references the object by index (for example, <code>document.myform.applets[3]</code>).</li><li>• Dreamweaver returns references for named objects contained in unnamed forms and layers (for example, <code>document.forms[2].myCheckbox</code>).</li></ul>
<b>Enabler</b>	None.

## History functions

History functions deal with undoing, redoing, recording, and playing steps that appear in the History palette. A step is any repeatable change to the document or to a selection in the document. Methods of the `dreamweaver.historyPalette` object either control or act on the selection in the History palette, not in the current document.

### `dom.redo()`

<b>Availability</b>	3.0
<b>Description</b>	Redoes the step that was just undone in the document.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dom.canRedo()</code>

### `dom.undo()`

<b>Availability</b>	3.0
<b>Description</b>	Undoes the previous step in the document.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dom.canUndo()</code>

### `dreamweaver.getRedoText()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the text associated with the editing operation that will be redone if the user chooses Edit > Redo or presses Ctrl+Y (Windows) or Command+Y (Macintosh).
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the text associated with the editing operation that will be redone.
<b>Enabler</b>	None.
<b>Example</b>	If the user's last action was make the selection bold, a call to <code>dw.getRedoText()</code> would return "Repeat Apply Bold".

### **dreamweaver.getUndoText()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the text associated with the editing operation that will be undone if the user chooses Edit > Undo or presses Ctrl+Z (Windows) or Command+Z (Macintosh).
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the text associated with the editing operation that will be undone.
<b>Enabler</b>	None.
<b>Example</b>	If the user's last action was to apply a CSS style to a selected range of text, a call to <code>dw.getUndoText()</code> would return "Undo Apply <span>".

### **dreamweaver.playRecordedCommand()**

<b>Availability</b>	3.0
<b>Description</b>	Plays the recorded command in the active document.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canPlayRecordedCommand()</code>

### **dreamweaver.redo()**

<b>Availability</b>	3.0
<b>Description</b>	Redoes the step that was just undone in the Document window, dialog box, floating palette, or Site window pane that has focus.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.canRedo()</code>

### **dreamweaver.startRecording()**

<b>Availability</b>	3.0
<b>Description</b>	Starts recording steps in the active document. The previously-recorded command is immediately discarded.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.isRecording()</code> (must return FALSE)

### **dreamweaver.stopRecording()**

<b>Availability</b>	3.0
<b>Description</b>	Stops recording without prompting the user.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.isRecording() (must return TRUE)

### **dreamweaver.undo()**

<b>Availability</b>	3.0
<b>Description</b>	Undoes the previous step in the Document window, dialog box, floating palette, or Site window pane that has focus.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.canUndo()

### **dreamweaver.historyPalette.clearSteps()**

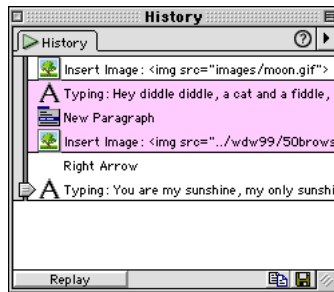
<b>Availability</b>	3.0
<b>Description</b>	Clears all steps from the History palette, and disables the Undo and Redo menu items.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.historyPalette.copySteps()**

<b>Availability</b>	3.0
<b>Description</b>	Copies the specified history steps to the clipboard. Dreamweaver warns the user of possible unintended consequences if the specified steps include an unrepeatable action.
<b>Arguments</b>	<i>arrayOfIndices</i>  The argument is an array of position indices in the History palette.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	The following code copies the first four steps in the History palette:  <code>dw.historyPalette.copySteps([0,1,2,3]);</code>

## `dreamweaver.historyPalette.getSelectedSteps()`

<b>Availability</b>	3.0
<b>Description</b>	Determines which portion of the History palette is selected.
<b>Arguments</b>	None.
<b>Returns</b>	An array containing the position indices of all of the selected steps.
<b>Enabler</b>	None.
<b>Example</b>	If the the second, third, and fourth steps are selected in the History palette, as shown, a call to <code>dw.historyPalette.getSelectedSteps()</code> would return <code>[1,2,3]</code> .

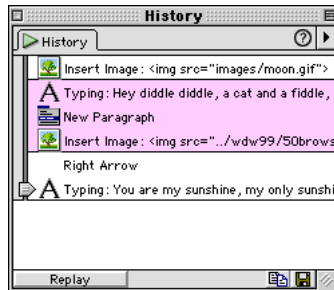


## `dreamweaver.historyPalette.getStepCount()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the number of steps in the History palette.
<b>Arguments</b>	None.
<b>Returns</b>	An integer representing the number of steps that are currently listed in the History palette.
<b>Enabler</b>	None.

## dreamweaver.historyPalette.getStepsAsJavaScript()

<b>Availability</b>	3.0
<b>Description</b>	Gets the JavaScript equivalent of the selected steps.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the JavaScript that corresponds to the selected steps.
<b>Enabler</b>	None.
<b>Example</b>	If the three steps shown in this example are selected in the History palette, a call to <code>dw.historyPalette.getStepsAsJavaScript(dw.historyPalette.getSelectedSteps())</code> returns <code>"dw.getDocumentDOM().insertText('Hey diddle diddle, a cat and a fiddle, the cow jumped over the moon.');"dw.getDocumentDOM().newBlock();\n</code> <code>dw.getDocumentDOM().insertHTML('&lt;img src=\"../wdw99/50browsers/images/sun.gif\"&gt;', true);\n"</code>



## dreamweaver.historyPalette.getUndoState()

<b>Availability</b>	3.0
<b>Description</b>	Gets the current undo state.
<b>Arguments</b>	None.
<b>Returns</b>	The position of the Undo marker in the History palette.
<b>Enabler</b>	None.



### **dreamweaver.historyPalette.replaySteps()**

<b>Availability</b>	3.0
<b>Description</b>	Replays the specified history steps in the active document. Dreamweaver warns the user of possible unintended consequences if the specified steps include an unrepeatable action.
<b>Arguments</b>	<i>arrayOfIndices</i>  The argument is an array of position indices in the History palette.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	A call to <code>dw.historyPalette.replaySteps([0,2,3])</code> plays the first, third, and fourth steps in the History palette.

### **dreamweaver.historyPalette.saveAsCommand()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Save As Command dialog box, which lets the user save the specified steps as a command. Dreamweaver warns the user of possible unintended consequences if the steps include an unrepeatable action.
<b>Arguments</b>	<i>arrayOfIndices</i>  The argument is an array of position indexes in the History palette.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	The following code saves the fourth, sixth, and eighth steps in the History palette as a command:  <code>dw.historyPalette.saveAsCommand([3,5,7]);</code>

### **dreamweaver.historyPalette.setSelectedSteps()**

<b>Availability</b>	3.0
<b>Description</b>	Selects the specified steps in the History palette.
<b>Arguments</b>	<i>arrayOfIndices</i>  The argument is an array of position indices in the History palette. If no argument is supplied, all steps are deselected.
<b>Returns</b>	None.
<b>Enabler</b>	None.
<b>Example</b>	The following code selects the first, second, and third steps in the History palette:  <code>dw.historyPalette.setSelection([0,1,2]);</code>

### **dreamweaver.historyPalette.setUndoState()**

<b>Availability</b>	3.0
<b>Description</b>	Performs the correct number of undo or redo operations to arrive at the specified undo state.
<b>Arguments</b>	<i>undoState</i>  The argument is the object returned by <code>dw.historyPalette.getUndoState()</code> .
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## **HTML style functions**

HTML style functions handle applying, creating, and deleting HTML styles. Methods of the `dreamweaver.htmlStylePalette` object either control or act on the selection in the HTML Style palette, not in the current document.

### **dom.applyHTMLStyle()**

<b>Availability</b>	3.0
<b>Description</b>	Applies the specified HTML style to the current selection. This function is valid only for the active document.
<b>Arguments</b>	<i>htmlStyleName</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.htmlStylePalette.deleteSelectedStyle()**

<b>Availability</b>	3.0
<b>Description</b>	Removes the selected style from the HTML Style palette.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.htmlStylePalette.canEditSelection()</code>

### **dreamweaver.htmlStylePalette.duplicateSelectedStyle()**

<b>Availability</b>	3.0
<b>Description</b>	Duplicates the selected style, and opens the Define HTML Style dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.htmlStylePalette.canEditSelection()

### **dreamweaver.htmlStylePalette.editSelectedStyle()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Define HTML Style dialog box for the selected style.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.htmlStylePalette.canEditSelection()

### **dreamweaver.htmlStylePalette.getSelectedStyle()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the name of the selected style in the HTML Style palette.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the name of the selected style.
<b>Enabler</b>	None.

### **dreamweaver.htmlStylePalette.getStyles()**

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of all of the names of the defined HTML styles.
<b>Arguments</b>	None.
<b>Returns</b>	An array of strings, each representing the name of an HTML style. If no HTML styles are defined, an empty array is returned.
<b>Enabler</b>	None.

### `dreamweaver.htmlStylePalette.newStyle()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the Define HTML Style dialog box for a new, untitled style.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.htmlStylePalette.setSelectedStyle()`

<b>Availability</b>	3.0
<b>Description</b>	Selects the specified style in the HTML Style palette.
<b>Arguments</b>	<i>htmlStyleName</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## Keyboard functions

Keyboard functions mimic document navigation tasks that are accomplished by pressing the arrow, Backspace, Delete, Page Up, and Page Down keys. In addition to such general arrow and key methods as `arrowLeft()` and `backspaceKey()`, Dreamweaver also provides methods for moving to the next or previous word or paragraph, and moving to the end of line or document or start of the line or document.

### `dom.arrowDown()`

<b>Availability</b>	3.0
<b>Description</b>	Moves the cursor down the specified number of times.
<b>Arguments</b>	<code>{nTimes}</code> , <code>{bShiftIsDown}</code> <ul style="list-style-type: none"><li>• The first argument is the number of times the cursor should be moved down. If this argument is omitted, the default is 1.</li><li>• The second argument is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is <code>FALSE</code>.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dom.arrowLeft()`

<b>Availability</b>	3.0
<b>Description</b>	Moves the cursor left the specified number of times.
<b>Arguments</b>	<code>{nTimes}</code> , <code>{bShiftIsDown}</code> <ul style="list-style-type: none"><li>• The first argument is the number of times the cursor should be moved left. If this argument is omitted, the default is 1.</li><li>• The second argument is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is <code>FALSE</code>.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### dom.arrowRight()

<b>Availability</b>	3.0
<b>Description</b>	Moves the cursor right the specified number of times.
<b>Arguments</b>	<p><code>{nTimes}</code>, <code>{bShiftIsDown}</code></p> <ul style="list-style-type: none"><li>• The first argument is the number of times the cursor should be moved right. If this argument is omitted, the default is 1.</li><li>• The second argument is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is <code>FALSE</code>.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### dom.arrowUp()

<b>Availability</b>	3.0
<b>Description</b>	Moves the cursor up the specified number of times.
<b>Arguments</b>	<p><code>{nTimes}</code>, <code>{bShiftIsDown}</code></p> <ul style="list-style-type: none"><li>• The first argument is the number of times the cursor should be moved up. If this argument is omitted, the default is 1.</li><li>• The second argument is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is <code>FALSE</code>.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### dom.backspaceKey()

<b>Availability</b>	3.0
<b>Description</b>	Equivalent to pressing the Backspace key the specified number of times. The exact behavior differs depending on whether there is a current selection or just an insertion point.
<b>Arguments</b>	<p><code>{nTimes}</code></p> <p>The argument is the number of times a Backspace operation should occur. If omitted, the default is 1.</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.deleteKey()**

<b>Availability</b>	3.0
<b>Description</b>	Equivalent to pressing the Delete key the specified number of times. The exact behavior differs depending on whether there is a current selection or just an insertion point.
<b>Arguments</b>	<i>{nTimes}</i>  The argument is the number of times a Delete operation should occur. If omitted, the default is 1.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.endOfDocument()**

<b>Availability</b>	3.0
<b>Description</b>	Moves the insertion point to the end of the document (that is, after the last visible content in the Document window, or after the closing HTML tag in the HTML inspector, depending on which window has focus).
<b>Arguments</b>	<i>{bShiftIsDown}</i>  The argument is a Boolean value indicating whether to extend the selection. If omitted, the default is FALSE.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.endOfLine()**

<b>Availability</b>	3.0
<b>Description</b>	Moves the insertion point to the end of the line.
<b>Arguments</b>	<i>{bShiftIsDown}</i>  The argument is a Boolean value indicating whether to extend the selection. If omitted, the default is FALSE.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### dom.nextParagraph()

<b>Availability</b>	3.0
<b>Description</b>	Moves the insertion point to the beginning of the next paragraph (or skips multiple paragraphs if <i>nTimes</i> is greater than 1).
<b>Arguments</b>	<i>{nTimes}</i> , <i>{bShiftIsDown}</i> <ul style="list-style-type: none"><li>• The first argument is the number of paragraphs ahead the cursor should jump. If this argument is omitted, the default is 1.</li><li>• The second argument is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is FALSE.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### dom.nextWord()

<b>Availability</b>	3.0
<b>Description</b>	Moves the insertion point to the beginning of the next word (or skips multiple words if <i>nTimes</i> is greater than 1).
<b>Arguments</b>	<i>{nTimes}</i> , <i>{bShiftIsDown}</i> <ul style="list-style-type: none"><li>• The first argument is the number of words ahead the cursor should jump. If this argument is omitted, the default is 1.</li><li>• The second argument is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is FALSE.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### dom.pageDown()

<b>Availability</b>	3.0
<b>Description</b>	Moves the insertion point down one page (equivalent to pressing Page Down).
<b>Arguments</b>	<i>{nTimes}</i> , <i>{bShiftIsDown}</i> <ul style="list-style-type: none"><li>• The first argument is the number of pages down the cursor should move. If this argument is omitted, the default is 1.</li><li>• The second argument is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is FALSE.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.



### dom.pageUp()

<b>Availability</b>	3.0
<b>Description</b>	Moves the insertion point up one page (equivalent to pressing Page Up).
<b>Arguments</b>	<p><i>{nTimes}</i>, <i>{bShiftIsDown}</i></p> <ul style="list-style-type: none"><li>• The first argument is the number of pages up the cursor should move. If this argument is omitted, the default is 1.</li><li>• The second argument is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is FALSE.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### dom.previousParagraph()

<b>Availability</b>	3.0
<b>Description</b>	Moves the insertion point to the beginning of the previous paragraph (or skips multiple paragraphs if <i>nTimes</i> is greater than 1).
<b>Arguments</b>	<p><i>{nTimes}</i>, <i>{bShiftIsDown}</i></p> <ul style="list-style-type: none"><li>• The first argument is the number of paragraphs back the cursor should jump. If this argument is omitted, the default is 1.</li><li>• The second argument is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is FALSE.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### dom.previousWord()

<b>Availability</b>	3.0
<b>Description</b>	Moves the insertion point to the beginning of the previous word (or skips multiple words if <i>nTimes</i> is greater than 1).
<b>Arguments</b>	<p><i>{nTimes}</i>, <i>{bShiftIsDown}</i></p> <ul style="list-style-type: none"><li>• The first argument is the number of words back the cursor should jump. If this argument is omitted, the default is 1.</li><li>• The second argument is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is FALSE.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.startOfDocument()

<b>Availability</b>	3.0
<b>Description</b>	Moves the insertion point to the beginning of the document (that is, before the first visible content in the Document window, or before the opening HTML tag in the HTML inspector, depending on which window has focus).
<b>Arguments</b>	<i>{bShiftIsDown}</i>  The argument is a Boolean value indicating whether to extend the selection. If omitted, the default is FALSE.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.startOfLine()

<b>Availability</b>	3.0
<b>Description</b>	Moves the insertion point to the beginning of the line.
<b>Arguments</b>	<i>{bShiftIsDown}</i>  The argument is a Boolean value indicating whether to extend the selection. If omitted, the default is FALSE.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## Layer and image map functions

Layer and image map functions handle aligning, resizing, and moving layers and image map hotspots. The function description indicates if it applies to layers or to hotspots.

### **dom.align()**

<b>Availability</b>	3.0
<b>Description</b>	Aligns the selected layers or hotspots left, right, top or bottom.
<b>Arguments</b>	<i>alignDirection</i>  The argument is the edge on which the layers or hotspots should be aligned—"left", "right", "top", or "bottom".
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canAlign()

### **dom.arrange()**

<b>Availability</b>	3.0
<b>Description</b>	Moves the selected hotspots in the specified direction.
<b>Arguments</b>	<i>toBackOrFront</i>  The argument is the direction in which the hotspots should be moved—"front" or "back".
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canArrange()

### **dom.makeSizesEqual()**

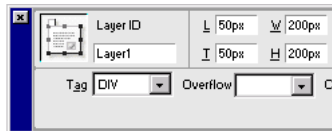
<b>Availability</b>	3.0
<b>Description</b>	Makes the selected layers or hotspots equal in height, width, or both. The last layer or hotspot selected is the guide.
<b>Arguments</b>	<i>bHoriz</i> , <i>bVert</i> <ul style="list-style-type: none"><li>• The first argument is a Boolean value indicating whether to resize the layers or hotspots horizontally.</li><li>• The second argument is a Boolean value indicating whether to resize the layers or hotspots vertically.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.moveSelectionBy()

<b>Availability</b>	3.0
<b>Description</b>	Moves the selected layers or hotspots by the specified number of pixels horizontally and vertically.
<b>Arguments</b>	<i>x, y</i> <ul style="list-style-type: none"><li>• The first argument is the number of pixels the selection should be moved horizontally.</li><li>• The second argument is the number of pixels the selection should be moved vertically.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.resizeSelectionBy()

<b>Availability</b>	3.0
<b>Description</b>	Resizes the currently selected layer or hotspot.
<b>Arguments</b>	<i>left, top, bottom, right</i> <ul style="list-style-type: none"><li>• The first argument is the new position of the left boundary of the layer or hotspot.</li><li>• The second argument is the new position of the top boundary of the layer or hotspot.</li><li>• The third argument is the new position of the bottom boundary of the layer or hotspot.</li><li>• The fourth argument is the new position of the right boundary of the layer or hotspot.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	If the selected layer has the Left, Top, Width, and Height properties shown, calling <code>dw.getDocumentDOM().resizeSelectionBy(-10,-30,30,10)</code> would be equivalent to resetting Left to 40, Top to 20, Width to 240, and Height to 240.



### dom.setLayerTag()

<b>Availability</b>	3.0
<b>Description</b>	Specifies the HTML tag that defines the selected layer or layers.
<b>Arguments</b>	<i>tagName</i>  The argument must be "layer", "ilayer", "div", or "span".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## Library and template functions

Library and template functions manage operations related to library items and templates, such as creating, updating, and breaking links between a document and a template or library item. Methods of the `dreamweaver.libraryPalette` object either control or act on the selection in the Library palette, not in the current document. Likewise, methods of the `dreamweaver.templatePalette` object control or act on the selection in the Template palette.

### dom.applyTemplate()

<b>Availability</b>	3.0
<b>Description</b>	Applies a template to the current document. If no argument is supplied, the Select Template dialog box appears. This function is valid only for the active document.
<b>Arguments</b>	<i>{templateURL}</i>  The argument is the path to a template in the current site, expressed as a file:// URL.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dom.canApplyTemplate()</code>

### dom.detachFromLibrary()

<b>Availability</b>	3.0
<b>Description</b>	Detaches the selected instance of a library item from its associated LBI file by removing the locking tags from around the selection. This function is equivalent to clicking Detach from Original in the Property inspector.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.detachFromTemplate()**

<b>Availability</b>	3.0
<b>Description</b>	Detaches the current document from its associated template.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.getAttachedTemplate()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the path of the template that is associated with the document.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the path of the template, expressed as a file:// URL.
<b>Enabler</b>	None.

### **dom.getEditableRegionList()**

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of all the editable regions in the body of the document.
<b>Arguments</b>	None.
<b>Returns</b>	An array of element nodes.
<b>Enabler</b>	None.
<b>Example</b>	See “dom.getSelectedEditableRegion()” on page 95.

### **dom.getIsLibraryDocument()**

<b>Availability</b>	3.0
<b>Description</b>	Determines whether the document is a library item.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the document is an LBI file.
<b>Enabler</b>	None.

### **dom.getIsTemplateDocument()**

<b>Availability</b>	3.0
<b>Description</b>	Determines whether the document is a template.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the document is a DWT file.
<b>Enabler</b>	None.

### **dom.getSelectedEditableRegion()**

<b>Availability</b>	3.0
<b>Description</b>	If the selection or insertion point is inside an editable region, gets the position of the editable region among all others in the body of the document.
<b>Arguments</b>	None.
<b>Returns</b>	An index into the array returned by dom.getEditableRegionList().
<b>Enabler</b>	None.
<b>Example</b>	The following code shows a dialog box containing the contents of the selected editable region:

```
var theDOM = dw.getDocumentDOM();
var edRegs = theDOM.getEditableRegionList();
var selReg = theDOM.getSelectedEditableRegion();
alert(edRegs[selReg].innerHTML);
```

### **dom.insertLibraryItem()**

<b>Availability</b>	3.0
<b>Description</b>	Inserts an instance of a library item into the document.
<b>Arguments</b>	<i>libraryItemURL</i>  The argument the path to an LBI file, expressed as a file:// URL.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.markSelectionAsEditable()**

<b>Availability</b>	3.0
<b>Description</b>	Displays the New Editable Region dialog box. When the user clicks New Region, Dreamweaver marks the selection as editable and leaves any text as is.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canMarkSelectionAsEditable()

### **dom.newEditableRegion()**

<b>Availability</b>	3.0
<b>Description</b>	Displays the New Editable Region dialog box. When the user clicks New Region, Dreamweaver inserts the name of the region, surrounded by curly braces, into the document at the cursor location.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canMakeNewEditableRegion()

### **dom.removeEditableRegion()**

<b>Availability</b>	3.0
<b>Description</b>	Removes an editable region from the document. If the editable region contains any content, the content is preserved—only the editable region markers are removed.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canRemoveEditableRegion()

### **dom.updateCurrentPage()**

<b>Availability</b>	3.0
<b>Description</b>	Updates the document's library items, templates, or both. This function is valid only for the active document.
<b>Arguments</b>	<i>{typeOfUpdate}</i>  The argument, if supplied, must be "library", "template", or "both". If omitted, the default is "both".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.



### **dreamweaver.updatePages()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Update Pages dialog box and selects the specified options.
<b>Arguments</b>	<i>{typeOfUpdate}</i>  The argument must be "library", "template", or "both". If the argument is omitted, it defaults to "both".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.libraryPalette.deleteSelectedItem()**

<b>Availability</b>	3.0
<b>Description</b>	Removes the selected library item from the Library palette and deletes its associated LBI file from the Library folder at the root of the current site. Instances of the deleted item may still exist in pages throughout the site.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.libraryPalette.getSelectedItem()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the path of the selected library item.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the path of the library item, expressed as a file:// URL.
<b>Enabler</b>	None.

### **dreamweaver.libraryPalette.newFromDocument()**

<b>Availability</b>	3.0
<b>Description</b>	Creates a new library item based on the selection in the current document.
<b>Arguments</b>	<i>bReplaceCurrent</i>  The argument is a Boolean value indicating whether to replace the selection with an instance of the newly-created library item.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.libraryPalette.recreateFromDocument()`

<b>Availability</b>	3.0
<b>Description</b>	Creates an LBI file for the selected instance of a library item in the current document. This function is equivalent to clicking Recreate in the Property inspector.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.libraryPalette.renameSelectedItem()`

<b>Availability</b>	3.0
<b>Description</b>	Turns the name of the selected library item into an edit field, allowing the user to rename the selection.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.templatePalette.deleteSelectedTemplate()`

<b>Availability</b>	3.0
<b>Description</b>	Deletes the selected template from the templates folder.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.templatePalette.getSelectedTemplate()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the path of the selected template.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the path of the template, expressed as a file:// URL.
<b>Enabler</b>	None.

### `dreamweaver.templatePalette.newBlankTemplate()`

<b>Availability</b>	3.0
<b>Description</b>	Creates a new template.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.templatePalette.renameSelectedTemplate()`

<b>Availability</b>	3.0
<b>Description</b>	Turns the name of the selected template into an edit field, allowing the user to rename the selection.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## Menu functions

Menu functions deal with optimizing and reloading the menus in Dreamweaver. The `dreamweaver.getMenuNeedsUpdating()` and `dreamweaver.notifyMenuUpdated()` functions are designed specifically to prevent unnecessary update routines from running on the dynamic menus that are built into Dreamweaver.

### `dreamweaver.getMenuNeedsUpdating()`

<b>Availability</b>	3.0
<b>Description</b>	Checks whether the specified menu needs to be updated.
<b>Arguments</b>	<i>menuId</i>  The argument is a string containing the value of the id attribute for the menu item (as specified in the menus.xml file).
<b>Returns</b>	A Boolean value indicating whether the menu needs to be updated. This function returns <code>FALSE</code> only if <code>dreamweaver.notifyMenuUpdated()</code> has been called with this <i>menuId</i> , and the return value of <i>menuListFunction</i> has not changed since that time. See “ <code>dreamweaver.notifyMenuUpdated()</code> ” on page 100 for more information.
<b>Enabler</b>	None.

## **dreamweaver.notifyMenuUpdated()**

<b>Availability</b>	3.0
<b>Description</b>	Notifies Dreamweaver when the specified menu needs to be updated.
<b>Arguments</b>	<i>menuId</i> , <i>menuListFunction</i> <ul style="list-style-type: none"><li>• The first argument is a string containing the value of the id attribute for the menu item (as specified in the menus.xml file).</li><li>• The second argument must be one of the following strings: "dw.cssStylePalette.getStyles()", "dw.getDocumentDOM().getFrameNames()", "dw.getDocumentDOM().getEditableRegionList", "dw.getBrowserList()", "dw.getRecentFileList()", "dw.getTranslatorList()", "dw.getFontList()", "dw.getDocumentList()", "dw.htmlStylePalette.getStyles()", or "site.getSites()".</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## **dreamweaver.reloadMenus()**

<b>Availability</b>	3.0
<b>Description</b>	Reloads the entire menu structure from the menus.xml file in the Configuration folder.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## Path functions

Path functions get and manipulate the paths to various files and folders on the user's hard drive. These functions determine the path to the root of the site in which the current document resides, convert relative paths to absolute URLs, and more.

### `dreamweaver.getConfigurationPath()`

<b>Availability</b>	2.0
<b>Description</b>	Gets the path to the Configuration folder, expressed as a file:// URL.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the path to the Configuration folder.
<b>Enabler</b>	None.
<b>Example</b>	<p>This function is useful when referencing other extension files, which are all stored in the Configuration folder inside the Dreamweaver application folder. For example:</p> <pre>var sortCmd = dreamweaver.getConfigurationPath() + "/Commands/Sort Table.htm" var sortDOM = dreamweaver.getDocumentDOM(sortCmd);</pre>

### `dreamweaver.getDocumentPath()`

<b>Availability</b>	1.2
<b>Description</b>	Gets the path of the specified document, expressed as a file:// URL. This function is equivalent to calling <code>dreamweaver.getDocumentDOM()</code> and then reading the URL property of the return value.
<b>Arguments</b>	<p><i>sourceDoc</i></p> <p>The argument must be "document", "parent", "parent.frames[number]", or "parent.frames[frameName]". "document" specifies the document that has the focus and contains the current selection. "parent" specifies the parent frameset (if the currently selected document is in a frame), and "parent.frames[number]" and "parent.frames[frameName]" specify a document that is in a particular frame within the frameset containing the current document.</p>
<b>Returns</b>	<p>One of the following values:</p> <ul style="list-style-type: none"><li>• A string containing the URL of the specified document if the file has been saved.</li><li>• An empty string if the file has not been saved.</li></ul>
<b>Enabler</b>	None.

## **dreamweaver.getSiteRoot()**

<b>Availability</b>	1.2
<b>Description</b>	Gets the local root folder (as specified in the Site Definition dialog box) for the site associated with the currently selected document, expressed as a file:// URL.
<b>Arguments</b>	None.
<b>Returns</b>	One of the following values: <ul style="list-style-type: none"><li>• A string containing the URL of the local root folder of the site within which the file has been saved.</li><li>• An empty string if the file is not associated with a site.</li></ul>
<b>Enabler</b>	None.

## **dreamweaver.relativeToAbsoluteURL()**

<b>Availability</b>	2.0
<b>Description</b>	Given a relative URL and a point of reference (either the path to the current document or the site root), converts the relative URL to an absolute (file://) URL.
<b>Arguments</b>	<i>relURL, docPath, siteRoot</i> <ul style="list-style-type: none"><li>• The first argument is the URL to be converted.</li><li>• The second argument is the path to a document on the user's disk (for example, the current document), expressed as a file:// URL, or an empty string if <i>relURL</i> is a root-relative URL.</li><li>• The third argument is the path to the site root, expressed as a file:// URL, or an empty string if <i>relURL</i> is a document-relative URL.</li></ul>
<b>Returns</b>	An absolute URL string. The return value is generated as follows: <ul style="list-style-type: none"><li>• If <i>relURL</i> is an absolute URL, no conversion takes place, and the return value is the same as <i>relURL</i>.</li><li>• If <i>relURL</i> is a document-relative URL, the return value is the combination of <i>docPath</i> + <i>relURL</i>.</li><li>• If <i>relURL</i> is a root-relative URL, the return value is the combination of <i>siteRoot</i> + <i>relURL</i>.</li></ul>
<b>Enabler</b>	None.

## Quick Tag Editor functions

Quick Tag Editor functions navigate through the tags within and surrounding the current selection. They remove any tag in the hierarchy, wrap the selection inside a new tag, and show the Quick Tag Editor to let the user edit specific attributes for the tag.

### `dom.selectChild()`

<b>Availability</b>	3.0
<b>Description</b>	Selects a child of the current selection. Calling this function is equivalent to selecting the next tag to the right in the tag selector at the bottom of the Document window.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dom.selectParent()`

<b>Availability</b>	3.0
<b>Description</b>	Selects the parent of the current selection. Calling this function is equivalent to selecting the next tag to the left in the tag selector at the bottom of the Document window.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dom.stripTag()`

<b>Availability</b>	3.0
<b>Description</b>	Removes the tag from around the current selection, leaving the contents, if any. If the selection contains more than one tag or no tags, Dreamweaver reports an error.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.wrapTag()

<b>Availability</b>	3.0
<b>Description</b>	Wraps the specified tag around the current selection. If the selection is unbalanced, Dreamweaver reports an error.
<b>Arguments</b>	<i>startTag</i>  The argument is the source associated with the opening tag.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	The following code wraps a link around the current selection: <pre>var theDOM = dw.getDocumentDOM(); var theSel = theDOM.getSelectedNode(); if (theSel.nodeType = Node.TEXT_NODE){     theDOM.wrapTag('&lt;a href="foo.html"&gt;'); }</pre>

## dreamweaver.showQuickTagEditor()

<b>Availability</b>	3.0
<b>Description</b>	Displays the Quick Tag Editor for the current selection.
<b>Arguments</b>	<i>{nearWhat}</i> , <i>{mode}</i> <ul style="list-style-type: none"><li>• The first argument, if specified, must be either "selection" or "tag selector". The default value if this argument is omitted is "selection".</li><li>• The second argument, if specified, must be "default", "wrap", "insert", or "edit". If <i>mode</i> is "default" or omitted, Dreamweaver uses heuristics to determine the mode to use for the current selection. <i>mode</i> is ignored if <i>nearWhat</i> is "tag selector".</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.



## Selection functions

Selection functions get and set the selection in open documents. For information on getting or setting the selection in the Site window, see “Site functions” on page 109.

### `dom.getSelectedNode()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the selected node. This function is equivalent to calling <code>dom.getSelection()</code> and then passing the return value to <code>dom.offsetsToNode()</code> .
<b>Arguments</b>	None.
<b>Returns</b>	The tag, text, or comment object that completely contains the specified range of characters.
<b>Enabler</b>	None.

### `dom.getSelection()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the selection, expressed as byte offsets into the document’s HTML source.
<b>Arguments</b>	<i>bAllowMultiple</i>  The argument is a Boolean value indicating whether the function should return multiple offsets if more than one table cell, image map hotspot, or layer is selected. If this argument is omitted, it defaults to FALSE.
<b>Returns</b>	For simple selections, an array containing two integers. The first integer is the byte offset of the beginning of the selection. The second integer is the byte offset of the end of the selection. If the two numbers are the same, then the current selection is an insertion point.  For complex selections (multiple table cells, multiple layers, or multiple image map hotspots), an array containing $2n$ integers, where $n$ is the number of selected items. The first integer in each pair is the byte offset of the beginning of the selection (including the opening TD, DIV, SPAN, LAYER, ILAYER, or MAP tag); the second integer in each pair is the byte offset of the end of the selection (including the closing TD, DIV, SPAN, LAYER, ILAYER, or MAP tag). If multiple table rows are selected, the offsets of each cell in each row are returned. The selection never includes the TR tags.
<b>Enabler</b>	None.

## dom.nodeToOffsets()

<b>Availability</b>	3.0
<b>Description</b>	Gets the position of a specific node in the DOM tree, expressed as byte offsets into the document's HTML source. Valid for any document on a local drive.
<b>Arguments</b>	<i>node</i>  The argument must be a tag, comment, or range of text that is a node in the tree returned by <code>dreamweaver.getDocumentDOM()</code> .
<b>Returns</b>	An array containing two integers. The first integer is the byte offset of the beginning of the tag, text, or comment; the second integer is the byte offset of the end of the node.
<b>Enabler</b>	None.
<b>Example</b>	The following code selects the first image object in the current document: <pre>var theDOM = dreamweaver.getDocumentDOM("document"); var theImg = theDOM.images[0]; var offsets = dreamweaver.nodeToOffsets(theImg); dreamweaver.setSelection(offsets[0], offsets[1]);</pre>

## dom.offsetsToNode()

<b>Availability</b>	3.0
<b>Description</b>	Gets the object in the DOM tree that completely contains the range of characters between the specified begin and end points. Valid for any document on a local drive.
<b>Arguments</b>	<i>offsetBegin, offsetEnd</i>  The arguments are the begin and end points, respectively, of a range of characters, expressed as byte offsets into the document's HTML source.
<b>Returns</b>	The tag, text, or comment object that completely contains the specified range of characters.
<b>Enabler</b>	None.
<b>Example</b>	The following code displays an alert if the selection is an image: <pre>var offsets = dreamweaver.getSelection(); var theSelection = dreamweaver.offsetsToNode(offsets[0], offsets[1]); if (theSelection.nodeType == Node.ELEMENT_NODE &amp;&amp; theSelection.tagName == 'IMG'){     alert('The current selection is an image.');</pre>

## dom.selectAll()

<b>Availability</b>	3.0
<b>Description</b>	Performs a Select All operation.

**Note:** In most cases this function selects all content in the active document. In some cases (for example, when the insertion point is inside a table), however, it selects only part of the active document. To set the selection to the entire document, use dom.setSelection().

<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dom.selectTable()

<b>Availability</b>	3.0
<b>Description</b>	Selects an entire table.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canSelectTable()

## dom.setSelectedNode()

<b>Availability</b>	3.0
<b>Description</b>	Sets the selected node. This function is equivalent to calling dom.nodeToOffsets() and then passing the return value to dom.setSelection().
<b>Arguments</b>	<i>node</i> , { <i>bSelectInside</i> }, { <i>bJumpToNode</i> } <ul style="list-style-type: none"><li>• The first argument is a text, comment, or element node in the document.</li><li>• The second argument is a Boolean value indicating whether to select the innerHTML of the node. This argument is relevant only if <i>node</i> is an element node, and it defaults to FALSE if omitted.</li><li>• The third argument is a Boolean value indicating whether to scroll the Document window, if necessary, to make the selection visible. If omitted, this argument defaults to FALSE.</li></ul>
<b>Returns</b>	Nothing
<b>Enabler</b>	None.

## dom.setSelection()

<b>Availability</b>	3.0
<b>Description</b>	Sets the selection in the document.
<b>Arguments</b>	<i>offsetBegin, offsetEnd</i>  The arguments are the begin and end points, respectively, for the new selection, expressed as byte offsets into the document's HTML source. If the two numbers are the same, the new selection is an insertion point. If the new selection is not a valid HTML selection, it is expanded to include the characters in the first valid HTML selection. For example, if <i>offsetBegin</i> and <i>offsetEnd</i> define the range SRC="myImage.gif" within <IMG SRC="myImage.gif">, the selection expands to include the entire IMG tag.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## dreamweaver.selectAll()

<b>Availability</b>	3.0
<b>Description</b>	Performs a Select All operation in the active document window or the Site window; or, on the Macintosh, the edit field that has focus in a dialog box or floating palette.  <b>Note:</b> If the operation takes place in the active document, in most cases it selects all content in the active document. In some cases (for example, when the insertion point is inside a table), however, it selects only part of the active document. To set the selection to the entire document, use dom.setSelection().
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.canSelectAll()

## Site functions

Site functions handle operations that affect files in the site files or site map. These functions create links between files; get, put, check in, and check out files; select and deselect files; create and remove files; get information about the sites that the user has defined; and more.

### `site.addToExistingFile()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the Select HTML File dialog box to let the user select a file, and then creates a link from the selected document to that file.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>site.canAddLink()</code>

### `site.addToNewFile()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the Link to New File dialog box to let the user specify details for the new file, and then creates a link from the selected document to that file.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>site.canAddLink()</code>

### `site.changeLinkSitewide()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the Change Link Sitewide dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### site.changeLink()

<b>Availability</b>	3.0
<b>Description</b>	Opens the Select HTML File dialog box to let the user select a new file for the link.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canChangeLink()

### site.checkIn()

<b>Availability</b>	3.0
<b>Description</b>	<p>Checks in the selected files, and handles dependent files in one of the following ways:</p> <ul style="list-style-type: none"><li>• If the user has selected Prompt on Put/Check In in the Site FTP preferences, the Dependent Files dialog box appears.</li><li>• If the user has previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked Yes, dependent files are uploaded and no dialog box appears.</li><li>• If the user has previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked No, dependent files are not uploaded and no dialog box appears.</li></ul>
<b>Arguments</b>	<p><i>siteOrURL</i></p> <p>The argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canCheckIn()

### site.checkLinks()

<b>Availability</b>	3.0
<b>Description</b>	Opens the Link Checker dialog box and checks links in the specified files.
<b>Arguments</b>	<p><i>scopeOfCheck</i></p> <p>The argument specifies where links will be checked. It must be "document", "selection", or "site".</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## site.checkOut()

<b>Availability</b>	3.0
<b>Description</b>	<p>Checks out the selected files, and handles dependent files in one of the following ways:</p> <ul style="list-style-type: none"><li>• If the user has selected Prompt on Get/Check Out in the Site FTP preferences, the Dependent Files dialog box appears.</li><li>• If the user has previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked Yes, dependent files are downloaded and no dialog box appears.</li><li>• If the user has previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked No, dependent files are not downloaded and no dialog box appears.</li></ul>
<b>Arguments</b>	<p><i>siteOrURL</i></p> <p>The argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canCheckOut()

## site.checkTargetBrowsers()

<b>Availability</b>	3.0
<b>Description</b>	Runs a target browser check on the selected files.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## site.defineSites()

<b>Availability</b>	3.0
<b>Description</b>	Opens the Define Sites dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **site.deleteSelection()**

<b>Availability</b>	3.0
<b>Description</b>	Deletes the selected files.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **site.locateInSite()**

<b>Availability</b>	3.0
<b>Description</b>	Locates the specified file or files in the specified pane of the Site window, and selects the found files.
<b>Arguments</b>	<i>localOrRemote, siteOrURL</i> <ul style="list-style-type: none"><li>• The first argument must be either "local" or "remote".</li><li>• The second argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canLocateInSite()

### **site.findLinkSource()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the file that contains the selected link or dependent file, and highlights the text of the link or the reference to the dependent in that file. This function operates only on files in the Site Map view.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canFindLinkSource()



## site.get()

<b>Availability</b>	3.0
<b>Description</b>	<p>Gets the specified files, and handles dependent files in one of the following ways:</p> <ul style="list-style-type: none"><li>• If the user has selected Prompt on Get/Check Out in the Site FTP preferences, the Dependent Files dialog box appears.</li><li>• If the user has at some point selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked Yes, dependent files are downloaded and no dialog box appears.</li><li>• If the user has at some point selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked No, dependent files are not downloaded and no dialog box appears.</li></ul>
<b>Arguments</b>	<p><i>siteOrURL</i></p> <p>The argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canGet()

## site.getCheckOutUser()

<b>Availability</b>	3.0
<b>Description</b>	Gets the login and check out name associated with the current site.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing a login and check out name, if defined, or an empty string if check in/check out is disabled.
<b>Enabler</b>	None.
<b>Example</b>	A call to site.getCheckOutUser() might return "lori (loriLaptop)". If no check out name is specified, only the login is returned (for example, "lori").

### site.getCheckOutUserForFile()

<b>Availability</b>	3.0
<b>Description</b>	Gets the login and check out name of the user that has the specified file checked out.
<b>Arguments</b>	<i>fileName</i>  The argument is the path to the file being inquired about, expressed as a file:// URL.
<b>Returns</b>	A string containing the login and check out name of the user that has the file checked out, or an empty string if the file is not checked out.
<b>Enabler</b>	None.
<b>Example</b>	A call to <code>site.getCheckOutUserForFile("file://C:/sites/avocado8/index.html")</code> might return "lori (loriLaptop)". If no check out name is specified, only the login is returned (for example, "lori").

### site.getConnectionState()

<b>Availability</b>	3.0
<b>Description</b>	Gets the current connection state.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the remote site is connected.
<b>Enabler</b>	<code>site.canConnect()</code>

### site.getCurrentSite()

<b>Availability</b>	3.0
<b>Description</b>	Gets the current site.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the name of the current site.
<b>Enabler</b>	None.
<b>Example</b>	If you have several sites defined, a call to <code>site.getCurrentSite()</code> returns the one that is currently showing in the Current Sites List in the Site window.

### **site.getFocus()**

<b>Availability</b>	3.0
<b>Description</b>	Determines which pane of the Site window has focus.
<b>Arguments</b>	None.
<b>Returns</b>	One of the following strings: "local", "remote", or "site map".
<b>Enabler</b>	None.

### **site.getLinkVisibility()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether all of the selected links in the site map are visible (that is, not marked hidden).
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether all of the selected links are visible.
<b>Enabler</b>	None.

### **site.getSelection()**

<b>Availability</b>	3.0
<b>Description</b>	Determines which files are currently selected in the Site window.
<b>Arguments</b>	None.
<b>Returns</b>	An array of strings representing the paths of the selected files and folders, expressed as file:// URLs, or an empty array if no files or folders are selected.
<b>Enabler</b>	None.

### **site.getSites()**

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of the defined sites.
<b>Arguments</b>	None.
<b>Returns</b>	An array of strings representing the names of the defined sites, or an empty array if no sites are defined.
<b>Enabler</b>	None.

### **site.invertSelection()**

<b>Availability</b>	3.0
<b>Description</b>	Inverts the selection in the site map.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **site.makeEditable()**

<b>Availability</b>	3.0
<b>Description</b>	Turns off the read-only flag on the selected files.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canMakeEditable()

### **site.makeNewDreamweaverFile()**

<b>Availability</b>	3.0
<b>Description</b>	Creates a new Dreamweaver file in the Site window (in the same directory as the first selected file or folder).
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canMakeNewFileOrFolder()

### **site.makeNewFolder()**

<b>Availability</b>	3.0
<b>Description</b>	Creates a new folder in the Site window (in the same directory as the first selected file or folder).
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canMakeNewFileOrFolder()

### **site.newHomePage()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the New Home Page dialog box to let the user create a new home page.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **site.newSite()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Site Definition Dialog box for a new, unnamed site.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **site.open()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the files that are currently selected in the Site window. If any folders are selected, they are expanded in the Site files view.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canOpen()

### site.put()

<b>Availability</b>	3.0
<b>Description</b>	<p>Puts the selected files, and handles dependent files in one of the following ways:</p> <ul style="list-style-type: none"><li>• If the user has selected Prompt on Put/Check In in the Site FTP preferences, the Dependent Files dialog box appears.</li><li>• If the user has previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked Yes, dependent files are uploaded and no dialog box appears.</li><li>• If the user has previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked No, dependent files are not uploaded and no dialog box appears.</li></ul>
<b>Arguments</b>	<p><i>siteOrURL</i></p> <p>The argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canPut()

### site.recreateCache()

<b>Availability</b>	3.0
<b>Description</b>	Recreates the cache for the current site.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canRecreateCache()

### site.refresh()

<b>Availability</b>	3.0
<b>Description</b>	Refreshes the file listing on the specified side of the Site window.
<b>Arguments</b>	<p><i>whichSide</i></p> <p>The argument must be "local", or "remote". If the site map has focus and <i>whichSide</i> is "local", the site map refreshes.</p>
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canRefresh()

### site.remotelsValid()

<b>Availability</b>	3.0
<b>Description</b>	Determines whether the remote site is valid.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether a remote site has been defined, and, if the server type is Local/Network, whether the drive is mounted.
<b>Enabler</b>	None.

### site.removeLink()

<b>Availability</b>	3.0
<b>Description</b>	Removes the selected link from the document above it in the site map.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canRemoveLink()

### site.renameSelection()

<b>Availability</b>	3.0
<b>Description</b>	Turns the name of the selected file into an edit field, allowing the user to rename the file. If more than one file is selected, this function acts on the last selected file.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### site.saveAsImage()

<b>Availability</b>	3.0
<b>Description</b>	Opens the Save As dialog box to let the user save the site map as an image.
<b>Arguments</b>	<i>fileType</i>  The argument is the type of image that should be saved. Valid values for Windows are "bmp" and "png"; valid values for Macintosh are "pict" and "jpeg". If the argument is omitted, or if the value is not valid on the current platform, the default is "bmp" in Windows and "pict" on the Macintosh.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **site.selectAll()**

<b>Availability</b>	3.0
<b>Description</b>	Selects all files in the active view (either the site map or the site files).
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **site.selectHomePage()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Open File dialog box to let the user choose a new home page.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **site.selectNewer()**

<b>Availability</b>	3.0
<b>Description</b>	Selects all files that are newer on the specified side of the Site window.
<b>Arguments</b>	<i>whichSide</i> The argument must be either "local" or "remote".
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canSelectNewer()

### **site.setAsHomePage()**

<b>Availability</b>	3.0
<b>Description</b>	Designates the file that is selected in the Site Files view as the home page for the site.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.



### site.setConnectionState()

<b>Availability</b>	3.0
<b>Description</b>	Sets the connection state of the current site.
<b>Arguments</b>	<i>bConnected</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### site.setCurrentSite()

<b>Availability</b>	3.0
<b>Description</b>	Opens the specified site in the local pane of the Site window.
<b>Arguments</b>	<i>whichSite</i>  The argument is the name of a defined site (as it appears in the Current Sites List in the Site window or the Define Sites dialog box).
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.
<b>Example</b>	If three sites are defined (for example, avocado8, dreamcentral, and testsite), a call to <code>site.setCurrentSite("dreamcentral");</code> makes dreamcentral the current site.

### site.setFocus()

<b>Availability</b>	3.0
<b>Description</b>	Gives focus to the specified pane of the Site window. If the specified pane is not showing, displays the pane and gives it focus.
<b>Arguments</b>	<i>whichPane</i>  The argument must be one of the following strings: "local", "remote", or "site map".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### site.setLayout()

<b>Availability</b>	3.0
<b>Description</b>	Opens the Site Map Layout pane of the Site Definition dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>site.canSetLayout()</code>

### site.setLinkVisibility()

<b>Availability</b>	3.0
<b>Description</b>	Shows or hides the current link.
<b>Arguments</b>	<i>bShow</i>  The argument is a Boolean value indicating whether to remove the Hidden designation from the current link.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### site.setSelection()

<b>Availability</b>	3.0
<b>Description</b>	Selects files or folders in the active pane in the Site window.
<b>Arguments</b>	<i>arrayOfURLs</i>  The argument is an array of strings, each a path to a file or folder in the current site, expressed as a file:// URL.  <b>Note:</b> Leave off the trailing slash (/) when specifying folder paths.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### site.synchronize()

<b>Availability</b>	3.0
<b>Description</b>	Opens the Synchronize Files dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canSynchronize()

### site.undoCheckOut()

<b>Availability</b>	3.0
<b>Description</b>	Removes the lock files that are associated with the selected files from the local and remote sites, and replaces the local copy of the selected files with the remote copy.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canUndoCheckOut()

## site.viewAsRoot()

<b>Availability</b>	3.0
<b>Description</b>	Temporarily moves the selected file to the top position in the Site map.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canViewAsRoot()

## String manipulation functions

String manipulation functions help you get information about a string, as well as convert a string from Latin 1 encoding to platform-native encoding and back.

## dreamweaver.getTokens()

<b>Availability</b>	1.0
<b>Description</b>	Accepts a string and splits it into tokens.
<b>Arguments</b>	<i>searchString</i> , <i>separatorCharacters</i> <ul style="list-style-type: none"><li>• The first argument is the string to be separated into tokens.</li><li>• The second argument is the character or characters that signifies the end of a token. Separator characters in quoted strings are ignored. If <i>separatorCharacters</i> contains a space, all white-space characters (for example, tabs) are treated as separator characters as if you had explicitly specified them. Two or more consecutive white space characters are treated as a single separator.</li></ul>
<b>Returns</b>	An array of token strings.
<b>Enabler</b>	None.
<b>Example</b>	<pre>dreamweaver.getTokens('foo("my arg 1", 34)', '(),,')</pre> returns the tokens: <ul style="list-style-type: none"><li>• foo</li><li>• "my arg 1"</li><li>• 34</li></ul>

## **dreamweaver.latin1ToNative()**

<b>Availability</b>	2.0
<b>Description</b>	Converts a string in Latin1 encoding to the native encoding on the user's machine. This function is intended for displaying the user interface of an extension file in another language.  <b>Note:</b> This function has no effect in Windows because Windows encodings are already based on Latin1.
<b>Arguments</b>	<i>stringToConvert</i>  The argument is the (already translated) string to be converted from Latin1 encoding to native encoding.
<b>Returns</b>	The converted string.
<b>Enabler</b>	None.

## **dreamweaver.nativeToLatin1()**

<b>Availability</b>	2.0
<b>Description</b>	Converts a string in native encoding to Latin1 encoding.  <b>Note:</b> This function has no effect in Windows because Windows encodings are already based on Latin1.
<b>Arguments</b>	<i>stringToConvert</i>  The argument is the string to be converted from native encoding to Latin1 encoding.
<b>Returns</b>	The converted string.
<b>Enabler</b>	None.

## Table editing functions

Table functions add and remove table rows and columns, change column widths and row heights, convert measurements from pixels to percents and back, and perform other standard table editing tasks.

### `dom.convertWidthsToPercent()`

<b>Availability</b>	3.0
<b>Description</b>	Converts all WIDTH attributes in the current table from pixels to percentages.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dom.convertWidthsToPixels()`

<b>Description</b>	Converts all WIDTH attributes in the current table from percentages to pixels.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dom.decreaseColspan()`

<b>Availability</b>	3.0
<b>Description</b>	Decreases the column span by 1.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dom.canDecreaseColspan()</code>

### `dom.decreaseRowspan()`

<b>Availability</b>	3.0
<b>Description</b>	Decreases the row span by 1.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dom.canDecreaseRowspan()</code>

### **dom.deleteTableColumn()**

<b>Availability</b>	3.0
<b>Description</b>	Removes the selected table column or columns.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canDeleteTableColumn()

### **dom.deleteTableRow()**

<b>Availability</b>	3.0
<b>Description</b>	Removes the selected table row or rows.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canDeleteTableRow()

### **dom.doDeferredTableUpdate()**

<b>Availability</b>	3.0
<b>Description</b>	If the Faster Table Editing option is selected in the General preferences, forces the table layout to reflect recent changes without moving the selection outside the table. This function has no effect if the Faster Table Editing option is not selected.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.getTableExtent()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the number of columns and rows in the selected table.
<b>Arguments</b>	None.
<b>Returns</b>	An array containing two whole numbers. The first array item is the number of columns, and the second array item is the number of rows. If no table was selected, nothing is returned.
<b>Enabler</b>	None.

### **dom.increaseColspan()**

<b>Availability</b>	3.0
<b>Description</b>	Increases the column span by 1.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canIncreaseColspan()

### **dom.increaseRowspan()**

<b>Availability</b>	3.0
<b>Description</b>	Increases the row span by 1.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canIncreaseRowspan()

### **dom.insertTableColumns()**

<b>Availability</b>	3.0
<b>Description</b>	Inserts the specified number of table columns into the current table.
<b>Arguments</b>	<i>numberOfCols, bBeforeSelection</i> <ul style="list-style-type: none"><li>• The first argument is the number of columns to insert.</li><li>• The second argument is a Boolean value indicating whether the columns should be inserted before the column that contains the selection.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canInsertTableColumns()

### **dom.insertTableRows()**

<b>Availability</b>	3.0
<b>Description</b>	Inserts the specified number of table rows into the current table.
<b>Arguments</b>	<i>numberOfRows, bBeforeSelection</i> <ul style="list-style-type: none"><li>• The first argument is the number of rows to insert.</li><li>• The second argument is a Boolean value indicating whether the rows should be inserted above the row that contains the selection.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canInsertTableRows()

### **dom.mergeTableCells()**

<b>Availability</b>	3.0
<b>Description</b>	Merges the selected table cells.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canMergeTableCells()

### **dom.removeAllTableHeights()**

<b>Availability</b>	3.0
<b>Description</b>	Removes all HEIGHT attributes from the selected table.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.removeAllTableWidths()**

<b>Availability</b>	3.0
<b>Description</b>	Removes all WIDTH attributes from the selected table.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setTableCellTag()**

<b>Availability</b>	3.0
<b>Description</b>	Specifies the tag for the selected cell.
<b>Arguments</b>	<i>tdOrTh</i> The argument must be either "td" or "th".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.



### **dom.setTableColumns()**

<b>Availability</b>	3.0
<b>Description</b>	Sets the number of columns in the selected table.
<b>Arguments</b>	<i>numberOfCols</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setTableRows()**

<b>Availability</b>	3.0
<b>Description</b>	Sets the number of rows in the selected table.
<b>Arguments</b>	<i>numberOfRows</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.showInsertTableRowsOrColumnsDialog()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Insert Rows or Columns dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canInsertTableColumns() or dom.canInsertTableRows()

### **dom.splitTableCell()**

<b>Availability</b>	3.0
<b>Description</b>	Splits the current table cell into the specified number of rows or columns. If one or both of the arguments is omitted, the Split Cells dialog box appears.
<b>Arguments</b>	<i>{colsOrRows}</i> , <i>{numberToSplitInto}</i> <ul style="list-style-type: none"><li>• The first argument, if supplied, must be either "columns" or "rows".</li><li>• The second argument, if supplied, is the number of rows or columns to split the cell into.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canSplitTableCell()

## Timeline functions

Timeline functions act on timelines. They add, remove, and change objects in a timeline; add behaviors, frames, and keyframes to a timeline; specify whether the timeline should play and loop automatically; and more. All of the functions in this section are methods of the `dreamweaver.timelineInspector` object because they affect the contents of the Timeline inspector.

### `dreamweaver.timelineInspector.addBehavior()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the Behavior inspector, and automatically supplies the correct <i>onFrameN</i> event (where <i>N</i> is the frame marked by the playback head) when the user chooses an action and clicks OK.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.timelineInspector.addFrame()`

<b>Availability</b>	3.0
<b>Description</b>	Adds a frame to the current timeline at the frame that contains the playback head.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.timelineInspector.canAddFrame()</code>

### `dreamweaver.timelineInspector.addKeyframe()`

<b>Availability</b>	3.0
<b>Description</b>	Adds a keyframe to the selected animation bar at the frame that contains the playback head.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.timelineInspector.canAddKeyFrame()</code>

### **dreamweaver.timelineInspector.addObject()**

<b>Availability</b>	3.0
<b>Description</b>	Adds the currently selected object to the timeline.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.timelineInspector.addTimeline()**

<b>Availability</b>	3.0
<b>Description</b>	Adds a new timeline to the current document.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.timelineInspector.changeObject()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Change Object dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.timelineInspector.canChangeObject()

### **dreamweaver.timelineInspector.getAutoplay()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the state of the Autoplay option for the current timeline.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the Autoplay option is selected.
<b>Enabler</b>	None.

### **dreamweaver.timelineInspector.getCurrentFrame()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current frame of the current timeline.
<b>Arguments</b>	None.
<b>Returns</b>	A frame number.
<b>Enabler</b>	None.

### **dreamweaver.timelineInspector.getLoop()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the state of the Loop option for the current timeline.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the Loop option is selected.
<b>Enabler</b>	None.

### **dreamweaver.timelineInspector.recordPathOfLayer()**

<b>Availability</b>	3.0
<b>Description</b>	Records the path of a layer as the user drags it.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.timelineInspector.removeBehavior()**

<b>Availability</b>	3.0
<b>Description</b>	Removes the selected behavior from the timeline.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dreamweaver.timelineInspector.canRemoveBehavior()

### **`dreamweaver.timelineInspector.removeFrame()`**

<b>Availability</b>	3.0
<b>Description</b>	Removes the selected frame from the timeline.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.timelineInspector.canRemoveFrame()</code>

### **`dreamweaver.timelineInspector.removeKeyframe()`**

<b>Availability</b>	3.0
<b>Description</b>	Removes the selected keyframe from an animation bar.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.timelineInspector.canRemoveKeyFrame()</code>

### **`dreamweaver.timelineInspector.removeObject()`**

<b>Availability</b>	3.0
<b>Description</b>	Removes the currently selected object from the timeline.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	<code>dreamweaver.timelineInspector.canRemoveObject()</code>

### **`dreamweaver.timelineInspector.removeTimeline()`**

<b>Availability</b>	3.0
<b>Description</b>	Removes the current timeline from the document.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.timelineInspector.renameTimeline()`

<b>Availability</b>	3.0
<b>Description</b>	Opens the Rename Timeline dialog box for the current timeline.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.timelineInspector.setAutoplay()`

<b>Availability</b>	3.0
<b>Description</b>	Sets the Autoplay option for the current timeline.
<b>Arguments</b>	<i>bAutoplay</i>  The argument is a Boolean value indicating whether to turn on the Autoplay option.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.timelineInspector.setCurrentFrame()`

<b>Availability</b>	3.0
<b>Description</b>	Moves the playback head to the specified frame.
<b>Arguments</b>	<i>frameNumber</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `dreamweaver.timelineInspector.setLoop()`

<b>Availability</b>	3.0
<b>Description</b>	Sets the Loop option for the current timeline.
<b>Arguments</b>	<i>bLoop</i>  The argument is a Boolean value indicating whether to turn on the Loop option.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## Toggle functions

Toggle functions get and set various options that are either on or off.

### `dom.getEditNoFramesContent()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the Modify > Frameset > Edit NoFrames Content option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the NOFRAMES content is the active view (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### `dom.getPreventLayerOverlaps()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the Prevent Layer Overlaps option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the option is on (TRUE) or off (FALSE).
<b>Enabler</b>	None.

### `dom.getShowFrameBorders()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Frame Borders option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether frame borders are visible (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### `dom.getShowGrid()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Grid > Show option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the grid is visible (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### **dom.getShowHeaderView()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Head Content option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether head content is visible (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### **dom.getShowImageMaps()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Image Maps option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether image maps are visible (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### **dom.getShowLayerBorders()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Layer Borders option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether layer borders are visible (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### **dom.getShowRulers()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Rulers > Show option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the rulers are visible (TRUE) or not (FALSE).
<b>Enabler</b>	None.



### **dom.getShowTableBorders()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Table Borders option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether table borders are visible (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### **dom.getShowTracingImage()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Tracing Image > Show option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the option is on (TRUE) or off (FALSE).
<b>Enabler</b>	None.

### **dom.getSnapToGrid()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Grid > Snap To option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether grid snapping is on (TRUE) or off (FALSE).
<b>Enabler</b>	None.

### **dom.setEditNoFramesContent()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the Modify > Frameset > Edit NoFrames Content option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bEditNoFrames</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canEditNoFramesContent()

### **dom.setPreventLayerOverlaps()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the Prevent Layer Overlaps option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bPreventLayerOverlaps</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setShowFrameBorders()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Frame Borders option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowFrameBorders</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setShowGrid()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Grid > Show option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowGrid</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setShowHeaderView()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Head Content option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowHead</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setShowImageMaps()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Image Maps option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowImageMaps</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setShowLayerBorders()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Layer Borders option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowLayerBorders</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setShowRulers()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Rulers > Show option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowRulers</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setShowTableBorders()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Table Borders option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowTableBorders</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setShowTracingImage()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Tracing Image > Show option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowTracingImage</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setSnapToGrid()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Grid > Snap To option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bSnapToGrid</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.getHideAllFloaters()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the Hide Floating Palettes option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the Hide Floating Palettes (TRUE) or the Show Floating Palettes (FALSE) menu item is available.
<b>Enabler</b>	None.

### **dreamweaver.getShowInvisibleElements()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Invisible Elements option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether invisible element markers are visible (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### **dreamweaver.getShowStatusBar()**

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the View > Status Bar option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the status bar is visible (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### **dreamweaver.setHideAllFloaters()**

<b>Availability</b>	3.0
<b>Description</b>	Turns on either the Hide Floating Palettes option (TRUE) or the Show Floating Palettes option (FALSE).
<b>Arguments</b>	<i>bShowFloatingPalettes</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.setShowInvisibleElements()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Invisible Elements option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bViewInvisibleElements</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.setShowStatusBar()**

<b>Availability</b>	3.0
<b>Description</b>	Turns the View > Status Bar option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowStatusBar</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### `site.getShowDependents()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the Show Dependent Files option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether dependent files are visible in the site map (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### `site.getShowHiddenFiles()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the Show Files Marked as Hidden option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether hidden files are visible in the site map (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### `site.getShowPageTitles()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the Show Page Titles option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether page titles are visible in the site map (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### `site.getShowToolTips()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the current state of the Tool Tips option.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether tool tips are visible in the Site window (TRUE) or not (FALSE).
<b>Enabler</b>	None.

### site.setShowDependents()

<b>Availability</b>	3.0
<b>Description</b>	Turns the Show Dependent Files option in the site map on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowDependentFiles</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### site.setShowHiddenFiles()

<b>Availability</b>	3.0
<b>Description</b>	Turns the Show Files Marked as Hidden option in the site map on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowHiddenFiles</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### site.setShowPageTitles()

<b>Availability</b>	3.0
<b>Description</b>	Turns the Show Page Titles option in the site map on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowPageTitles</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	site.canShowPageTitles()

### site.setShowToolTips()

<b>Availability</b>	3.0
<b>Description</b>	Turns the Tool Tips option on (TRUE) or off (FALSE).
<b>Arguments</b>	<i>bShowToolTips</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## Translation functions

Translation functions either deal directly with translators or with the results of translation. These functions get information about or run a translator, edit content in a locked region, and specify that the translated source should be used when getting and setting selection offsets.

### **dom.runTranslator()**

<b>Availability</b>	3.0
<b>Description</b>	Runs the specified translator on the document. This function is valid only for the active document.
<b>Arguments</b>	<i>translatorName</i>  The argument is the name of a translator as it appears in the Translation preferences.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.editLockedRegions()**

<b>Availability</b>	2.0
<b>Description</b>	Depending on the value of the argument, makes locked regions editable or noneditable. By default, locked regions are noneditable; if you try to edit a locked region before specifically making it editable with this function, Dreamweaver beeps and disallows the change.  <b>Note:</b> Editing locked regions can have unintended consequences for library items and templates. It is not recommend that you use this function outside the context of data translators.
<b>Arguments</b>	<i>bAllowEdits</i>  The argument is a Boolean value indicating that edits are allowed (TRUE) or not allowed (FALSE). Dreamweaver automatically restores locked regions to their default (noneditable) state when the script that calls this function finishes executing.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.



## **dreamweaver.getTranslatorList()**

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of the installed translators.
<b>Arguments</b>	None.
<b>Returns</b>	An array of strings, each representing the name of a translator as it appears in the Translation preferences.
<b>Enabler</b>	None.

## **dreamweaver.useTranslatedSource()**

<b>Availability</b>	2.0
<b>Description</b>	Specifies that the values returned by <code>dreamweaver.nodeToOffsets()</code> and <code>dreamweaver.getSelection()</code> and used by <code>dreamweaver.offsetsToNode()</code> and <code>dreamweaver.setSelection()</code> should be offsets into the translated source (the HTML contained in the DOM after a translator is run), not the untranslated source.  <b>Note:</b> This function is only relevant in property inspector files.
<b>Arguments</b>	<i>bUseTranslatedSource</i>  The default value of the argument is <code>FALSE</code> . Dreamweaver automatically uses the untranslated source for subsequent calls to <code>dw.getSelection()</code> , <code>dw.setSelection()</code> , <code>dw.nodeToOffsets()</code> , and <code>dw.offsetsToNode()</code> when the script that calls <code>dw.useTranslatedSource()</code> finishes executing, if <code>dw.useTranslatedSource()</code> is not explicitly called with an argument of <code>FALSE</code> before then.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## Visual layout functions

Visual layout functions handle operations that affect the layout environment. They affect the source, position, and opacity of the tracing image; get and set the ruler origin and units; turn the grid on and off and change its settings; and play or stop playing plugins.

### `dom.getRulerOrigin()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the origin of the ruler.
<b>Arguments</b>	None.
<b>Returns</b>	An array of two integers. The first array item is the x coordinate of the origin, and the second array item is the y coordinate of the origin. Both values are in pixels.
<b>Enabler</b>	None.

### `dom.getRulerUnits()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the current ruler units.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing one of the following values: <ul style="list-style-type: none"><li>• "in"</li><li>• "cm"</li><li>• "px"</li></ul>
<b>Enabler</b>	None.

### `dom.getTracingImageOpacity()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the opacity setting for the document's tracing image.
<b>Arguments</b>	None.
<b>Returns</b>	A value between 0 and 100, or nothing if no opacity is set.
<b>Enabler</b>	<code>dom.hasTracingImage()</code>

### **dom.loadTracingImage()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Select Image Source dialog box. If the user selects an image and clicks OK, the Page Properties dialog box opens with the Tracing Image field filled in.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.playAllPlugins()**

<b>Availability</b>	3.0
<b>Description</b>	Plays all plugin content in the document.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.playPlugin()**

<b>Availability</b>	3.0
<b>Description</b>	Plays the selected plugin item.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.canPlayPlugin()

### **dom.setRulerOrigin()**

<b>Availability</b>	3.0
<b>Description</b>	Sets the origin of the ruler.
<b>Arguments</b>	<i>xCoordinate, yCoordinate</i> <ul style="list-style-type: none"><li>• The first argument is a value, expressed in pixels, on the horizontal axis.</li><li>• The second argument is a value, expressed in pixels, on the vertical axis.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setRulerUnits()**

<b>Availability</b>	3.0
<b>Description</b>	Sets the current ruler units.
<b>Arguments</b>	<i>units</i> The argument must be "px", "in", or "cm".
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.setTracingImagePosition()**

<b>Availability</b>	3.0
<b>Description</b>	Moves the top left corner of the tracing image to the specified coordinates. If the arguments are omitted, the Page Properties dialog box appears.
<b>Arguments</b>	<i>x, y</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.hasTracingImage()

### **dom.setTracingImageOpacity()**

<b>Availability</b>	3.0
<b>Description</b>	Sets the opacity of the tracing image.
<b>Arguments</b>	<i>opacityPercentage</i> The argument must be a number between 0 and 100.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.hasTracingImage()
<b>Example</b>	The following code sets the opacity of the tracing image to 30%: <code>dw.getDocumentDOM().setTracingOpacity('30');</code>

### **dom.snapTracingImageToSelection()**

<b>Availability</b>	3.0
<b>Description</b>	Aligns the top left corner of the tracing image with the top left corner of the current selection.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	dom.hasTracingImage()

### **dom.stopAllPlugins()**

<b>Availability</b>	3.0
<b>Description</b>	Stops all plugin content that is currently playing in the document.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dom.stopPlugin()**

<b>Availability</b>	3.0
<b>Description</b>	Stops the selected plugin item.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection is currently being played with a plugin.
<b>Enabler</b>	dom.canStopPlugin()

### **dreamweaver.arrangeFloatingPalettes()**

<b>Availability</b>	3.0
<b>Description</b>	Moves the visible floating palettes to their default positions.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

### **dreamweaver.showGridSettingsDialog()**

<b>Availability</b>	3.0
<b>Description</b>	Opens the Grid Settings dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## Window functions

Window functions deal with operations related to the Document window and the floating palettes. These functions show and hide floating palettes, determine which part of the Document window has focus, and set the active document. For operations related specifically to the Site window, see “Site functions” on page 109.

### `dom.getFocus()`

<b>Availability</b>	3.0
<b>Description</b>	Determines the part of the document that has focus.
<b>Arguments</b>	None.
<b>Returns</b>	One of the following strings: <ul style="list-style-type: none"><li>• "head" if the HEAD area is active</li><li>• "body" if the BODY or NOFRAMES area is active</li><li>• "frameset" if a frameset or any of its frames is selected</li><li>• "none" if the focus is not in the document (for example, if it's in the Property inspector or another floating palette)</li></ul>
<b>Enabler</b>	None.

### `dom.getWindowTitle()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the title of the window that contains the document.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the text that appears between the TITLE tags in the document, or nothing if the document is not in an open window.
<b>Enabler</b>	None.

### `dreamweaver.getActiveWindow()`

<b>Availability</b>	3.0
<b>Description</b>	Gets the document in the active window.
<b>Arguments</b>	None.
<b>Returns</b>	The document object corresponding to the document in the active window; or, if the document is in a frame, the document object corresponding to the frameset.
<b>Enabler</b>	None.

## **dreamweaver.getDocumentList()**

<b>Availability</b>	3.0
<b>Description</b>	Gets a list of all the open documents.
<b>Arguments</b>	None.
<b>Returns</b>	An array of document objects, each corresponding to an open Document window. If a Document window contains a frameset, the document object refers to the frameset, not the contents of the frames.
<b>Enabler</b>	None.

## **dreamweaver.getFocus()**

<b>Availability</b>	3.0
<b>Description</b>	Determines the part of the application that has focus.
<b>Arguments</b>	<i>bAllowFloaters</i>
<b>Returns</b>	One of the following strings: <ul style="list-style-type: none"><li>• "document" if the Document window is active</li><li>• "site" if the Site window is active</li><li>• <i>floaterName</i> if <i>bAllowFloaters</i> is TRUE and a floating palette has focus, where <i>floaterName</i> is "objects", "properties", "launcher", "library", "css styles", "html styles", "behaviors", "timelines", "html", "layers", "frames", "templates", or "history"</li><li>• (Macintosh) "none" if neither the Site window nor any Document windows are open</li></ul>
<b>Enabler</b>	None.

## **dreamweaver.getFloaterVisibility()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether the specified palette or inspector is visible.
<b>Arguments</b>	<i>floaterName</i>  The argument is the name of a floating palette. The built-in palettes must be referenced using one the following strings: "objects", "properties", "launcher", "site files", "site map", "library", "css styles", "html styles", "behaviors", "timelines", "html", "layers", "frames", "templates", or "history". If <i>floaterName</i> does not match one of the built-in palette names, Dreamweaver searches in the Configuration/Floaters folder for a file called <i>floaterName</i> .htm.
<b>Returns</b>	TRUE if the floater is visible and frontmost, FALSE otherwise or if Dreamweaver cannot find a floater with the name <i>floaterName</i> .
<b>Enabler</b>	None.

## **dreamweaver.setActiveWindow()**

<b>Availability</b>	3.0
<b>Description</b>	Activates the window containing the specified document.
<b>Arguments</b>	<i>documentObject</i> , { <i>bActivateFrame</i> }
	<ul style="list-style-type: none"><li>• The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code>).</li><li>• The second argument, applicable only if <i>documentObject</i> is inside a frameset, is a Boolean value indicating whether to activate the frame that contains the document as well as the window that contains the frameset.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## **dreamweaver.setFloaterVisibility()**

<b>Availability</b>	3.0
<b>Description</b>	Specifies whether to make a particular floating palette or inspector visible.
<b>Arguments</b>	<i>floaterName</i> , <i>bIsVisible</i>
	<ul style="list-style-type: none"><li>• The argument is the name of a floating palette. The built-in palettes must be referenced using one of the following strings: "objects", "properties", "launcher", "site files", "site map", "library", "css styles", "html styles", "behaviors", "timelines", "html", "layers", "frames", "templates", or "history". If <i>floaterName</i> does not match one of the built-in palette names, Dreamweaver searches in the Configuration/Floaters folder for a file called <i>floaterName</i>.htm. If Dreamweaver cannot find a floater with the name <i>floaterName</i>, this function has no effect.</li><li>• The second argument is a Boolean value indicating whether to make the floater visible.</li></ul>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## **dreamweaver.showProperties()**

<b>Availability</b>	3.0
<b>Description</b>	Makes the Property inspector visible and gives it focus.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.



## **dreamweaver.toggleFloater()**

<b>Availability</b>	3.0
<b>Description</b>	Shows, hides, or brings to the front the specified palette or inspector.  <b>Note:</b> This function is only meaningful in the menus.xml file. To show, bring forward, or hide a floater, use <code>dw.setFloaterVisibility()</code> .
<b>Arguments</b>	<i>floaterName</i>
<b>Returns</b>	Nothing.
<b>Enabler</b>	None.

## Deprecated functions

Deprecated functions work, but have been superceded by newer features in Dreamweaver. You should use the newer alternatives, because support for the deprecated functions may be withdrawn in future Dreamweaver versions.

## **dreamweaver.getBehaviorEvent()**

<b>Availability</b>	1.2, deprecated in 2.0 because actions are now chosen before events.
<b>Description</b>	In a behavior action file, gets the event that triggers this action.
<b>Arguments</b>	None.
<b>Returns</b>	A string representing the event. This is the same string that is passed as an argument ( <i>event</i> ) to the <code>canAcceptBehavior()</code> function.
<b>Example</b>	<p>The following instance of <code>getBehaviorEvent()</code> is from the <code>initializeUI()</code> function in the Drag Layer action file. The role of the following snippet is similar to that of <code>canAcceptBehavior()</code>—that is, it checks whether the selected event is appropriate for the selected action. Such a construct is more useful than <code>canAcceptBehavior()</code>, because it lets you offer the user some guidance about which events are suitable for calling the selected action. <code>canAcceptBehavior()</code> can only make the action unavailable in the Actions pop-up menu if the user chooses an inappropriate event.</p> <pre>theEvent = dreamweaver.getBehaviorEvent().toLowerCase(); CANBEAPPLIED = (theEvent != "onmousedown" &amp;&amp; theEvent != "onmousemove"); if (CANBEAPPLIED) {     [display the Drag Layer UI] } else{     [display a helpful message that tells the user which events are appropriate for     the Drag Layer action.] }</pre>

## **dreamweaver.getObjectRefs()**

<b>Availability</b>	1.0
<b>Description</b>	Scans the specified documents for instances of the specified tags (or, if no tags are specified, for all tags in the document) and formulates browser-specific references to them. This function is equivalent to calling <code>getElementsByTagName()</code> and then calling <code>dreamweaver.getElementRef()</code> for each tag in the nodelist.
<b>Arguments</b>	<p><i>NSorIE, sourceDoc, {tag1}, {tag2},... {tagN}</i></p> <ul style="list-style-type: none"><li>• The first argument must be either "NS 4.0" or "IE 4.0". The DOM and rules for nested references differ in Navigator 4.0 and Internet Explorer 4.0. This argument specifies for which browser to return a valid reference.</li><li>• The second argument must be "document", "parent", "parent.frames[number]", "parent.frames['frameName']", or a URL. document specifies the document that has the focus and contains the current selection. parent specifies the parent frameset (if the currently selected document is in a frame), and parent.frames[number] and parent.frames['frameName'] specify a document that is in a particular frame within the frameset containing the current document. If the argument is a relative URL, it is relative to the extension file.</li><li>• The third and subsequent arguments, if supplied, are the names of tags (for example, "IMG", "FORM", "HR").</li></ul>
<b>Returns</b>	<p>An array of strings, each a valid JavaScript reference to a named instance of the requested tag type in the specified document (for example, "document.myLayer.document.myImage") for the specified browser.</p> <ul style="list-style-type: none"><li>• Dreamweaver returns correct references for Internet Explorer for A, AREA, APPLET, EMBED, DIV, SPAN, INPUT, SELECT, OPTION, TEXTAREA, OBJECT, and IMG tags.</li><li>• Dreamweaver returns correct references for Navigator for A, AREA, APPLET, EMBED, LAYER, ILAYER, SELECT, OPTION, TEXTAREA, OBJECT, and IMG tags, and for absolutely positioned DIV and SPAN tags. For DIV and SPAN tags that are not absolutely positioned, Dreamweaver returns "cannot reference &lt;tag&gt;".</li><li>• Dreamweaver does not return references for unnamed objects. If an object does not contain either a NAME or an ID attribute, then Dreamweaver returns "unnamed &lt;tag&gt;". If the browser does not support a reference by name, Dreamweaver references the object by index (for example, document.myform.applets[3]).</li><li>• Dreamweaver does return references for named objects contained in unnamed forms and layers (for example, document.forms[2].myCheckbox).</li></ul>

When the same list of arguments is passed to `getObjectTags()`, the two functions return arrays of the same length and with parallel content.

**Example** `dreamweaver.getObjectRefs("NS 4.0", "document", "IMG")`, depending on the contents of the active document, might return an array with the following items:

- `"document.bullet"`
- `"document.layers['headerLayer'].document.header"`
- `"document.photoLayer.document.headshot"`

## **dreamweaver.getObjectTags()**

**Availability** 1.0

**Description** Scans the specified document for instances of the specified tags or, if no tags are specified, for all tags in the document. This function is equivalent to calling `getElementsByTagName()` and then getting `outerHTML` for each element in the nodelist.

**Arguments** `sourceDoc`, `{tag1}`, `{tag2}`,... `{tagN}`

- The first argument must be `"document"`, `"parent"`, `"parent.frames[number]"`, `"parent.frames['frameName']"`, or a URL. `document` specifies the document that has the focus and contains the current selection. `parent` specifies the parent frameset (if the currently selected document is in a frame), and `parent.frames[number]` and `parent.frames['frameName']` specify a document that is in a particular frame within the frameset containing the current document. If the argument is a relative URL, it is relative to the extension file.
- The second and subsequent arguments, if supplied, are the names of tags (for example, `"IMG"`, `"FORM"`, `"HR"`).

**Returns** An array of strings, each the HTML source for an instance of the requested tag type in the specified document.

- If one of the `tag` arguments is `LAYER`, the function returns all `LAYER` and `ILAYER` tags and all absolutely positioned `DIV` and `SPAN` tags.
- If one of the `tag` arguments is `INPUT`, the function returns all form elements. To get a particular type of form element, specify `INPUT/TYPE`, where `TYPE` is `button`, `text`, `radio`, `checkbox`, `password`, `textarea`, `select`, `hidden`, `reset`, or `submit`.

When the same list of arguments is passed to `getObjectRefs()`, the two functions return arrays of the same length.

**Example** `dreamweaver.getObjectTags("document", "IMG")`, depending on the contents of the active document, might return an array with the following items:

- `'<IMG SRC="/images/dot.gif" WIDTH="10" HEIGHT="10" NAME="bullet">'`
- `'<IMG SRC="header.gif" WIDTH="400" HEIGHT="32" NAME="header">'`
- `'<IMG SRC="971208_nj.jpg" WIDTH="119" HEIGHT="119" NAME="headshot">'`

## **dreamweaver.getSelection()**

<b>Availability</b>	2.0, deprecated in 3.0 in favor of <code>dom.getSelection()</code> .
<b>Description</b>	Gets the selection in the current document, expressed as byte offsets into the document's HTML source.
<b>Arguments</b>	None.
<b>Returns</b>	An array containing two integers. The first integer is the byte offset of the beginning of the selection; the second integer is the byte offset of the end of the selection. If the two numbers are the same, then the current selection is an insertion point.

## **dreamweaver.nodeToOffsets()**

<b>Availability</b>	2.0, deprecated in 3.0 in favor of <code>dom.nodeToOffsets()</code> .
<b>Description</b>	Gets the position of a specific node in the DOM tree, expressed as byte offsets into the document's HTML source.
<b>Arguments</b>	<i>node</i>  The argument must be a tag, comment, or range of text that is a node in the tree returned by <code>dreamweaver.getDocumentDOM()</code> .
<b>Returns</b>	An array containing two integers. The first integer is the byte offset of the beginning of the tag, text, or comment; the second integer is the byte offset of the end of the node.
<b>Example</b>	The following code selects the first image object in the current document: <pre>var theDOM = dreamweaver.getDocumentDOM("document"); var theImg = theDOM.images[0]; var offsets = dreamweaver.nodeToOffsets(theImg); dreamweaver.setSelection(offsets[0], offsets[1]);</pre>

## **dreamweaver.offsetsToNode()**

<b>Availability</b>	2.0, deprecated in 3.0 in favor of <code>dom.offsetsToNode()</code> .
<b>Description</b>	Gets the object in the DOM tree that completely contains the range of characters between the specified begin and end points.
<b>Arguments</b>	<i>offsetBegin, offsetEnd</i>  The arguments are the begin and end points, respectively, of a range of characters, expressed as byte offsets into the document's HTML source.
<b>Returns</b>	The tag, text, or comment object that completely contains the specified range of characters.
<b>Example</b>	The following code displays an alert if the selection is an image: <pre>var offsets = dreamweaver.getSelection(); var theSelection = dreamweaver.offsetsToNode(offsets[0], offsets[1]); if (theSelection.nodeType == Node.ELEMENT_NODE &amp;&amp; theSelection.tagName == 'IMG'){     alert('The current selection is an image.');</pre>

## **dreamweaver.popupCommand()**

<b>Availability</b>	2.0, deprecated in 3.0 in favor of <code>dreamweaver.runCommand()</code> .
<b>Description</b>	Executes the specified command. To the user, the effect is the same as choosing the command from a menu; if a dialog box is associated with the command, it appears. This function provides the ability to call a command from another extension file. It blocks other edits until the user dismisses the dialog box.  <b>Note:</b> This function can be called only within <code>objectTag()</code> or in any script in a command or property inspector file.
<b>Arguments</b>	<i>commandFile</i>  The argument is the name of a command file within the Configuration/Commands folder (for example, "Format Table.htm").
<b>Returns</b>	Nothing.

## **dreamweaver.setSelection()**

<b>Availability</b>	2.0, deprecated in 3.0 in favor of <code>dom.setSelection()</code> .
<b>Description</b>	Sets the selection in the current document. This function can move the selection only within the current document; it cannot change the focus to a different document.
<b>Arguments</b>	<i>offsetBegin</i> , <i>offsetEnd</i>  The arguments are the begin and end points, respectively, for the new selection, expressed as byte offsets into the document's HTML source. If the two numbers are the same, the new selection is an insertion point. If the new selection is not a valid HTML selection, it is expanded to include the characters in the first valid HTML selection. For example, if <i>offsetBegin</i> and <i>offsetEnd</i> define the range <code>SRC="myImage.gif"</code> within <code>&lt;IMG SRC="myImage.gif"&gt;</code> , the selection expands to include the entire <code>IMG</code> tag.
<b>Returns</b>	Nothing.

## **Enablers**

Enabler functions determine whether to enable menu items based on whether Dreamweaver can perform specific operations in the current context. The function specifications describe the general circumstances under which each function returns `TRUE`. However, the descriptions are not intended to be comprehensive and may exclude some cases in which the function would return `FALSE`.

### **dom.canAlign()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Align Left, Align Right, Align Top, or Align Bottom operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether two or more layers or hotspots are selected.

### **dom.canApplyTemplate()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Apply To Page operation. This function is valid only for the active document.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the document is not a library item or a template, and that the selection is not within the <code>NOFRAMES</code> tag.

### **dom.canArrange()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Bring to Front or Move to Back operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether a hotspot is selected.

### **dom.canClipCopyText()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Copy as Text operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection is a range (that is, not an insertion point).

### **dom.canClipPaste()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Paste operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the clipboard contains any content that can be pasted into Dreamweaver.

### **dom.canClipPasteText()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Paste as Text operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the clipboard contains any content that can be pasted into Dreamweaver as text.

### **dom.canConvertLayersToTable()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Convert Layers to Table operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether all content in the BODY of the document is contained within layers.

### **dom.canConvertTablesToLayers()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Convert Tables to Layers operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether all the content in the BODY of the document is contained within tables, and the document is not based on a template.

### **dom.canDecreaseColspan()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Decrease Colspan operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the current cell has a COLSPAN attribute, and whether that attribute's value is greater than or equal to 2.

### **dom.canDecreaseRowspan()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Decrease Rowspan operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the current cell has a ROWSPAN attribute, and whether that attribute's value is greater than or equal to 2.

### **dom.canDeleteTableColumn()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Delete Column operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether a the insertion point is inside a cell, or if a cell or column is selected.

### **dom.canDeleteTableRow()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Delete Row operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether a the insertion point is inside a cell, or if a cell or row is selected.



### **dom.canEditNoFramesContent()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Edit NoFrames Content operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the current document is a frameset or within a frameset.

### **dom.canIncreaseColspan()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Increase Colspan operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether there are any cells to the right of the current cell.

### **dom.canIncreaseRowspan()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Increase Rowspan operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether there are any cells below the current cell.

### **dom.canInsertTableColumns()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Insert Column(s) operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection is inside a table. This function returns FALSE if the selection is an entire table.

### **dom.canInsertTableRows()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Insert Row(s) operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection is inside a table. This function returns FALSE if the selection is an entire table.

### **dom.canMakeNewEditableRegion()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a New Editable Region operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the current document is a template (.dwt) file.

### **dom.canMarkSelectionAsEditable()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Mark Selection as Editable operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether there is a selection, and whether the current document is a template (.dwt) file.

### **dom.canMergeTableCells()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Merge Cells operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection is a rectangular grouping of table cells.

### **dom.canPlayPlugin()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Play operation. This function is valid only for the active document.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection can be played with a plugin.

### **dom.canRedo()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Redo operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether any steps remain to redo.

### **dom.canRemoveEditableRegion()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Unmark Editable Region operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the current document is a template.

### **dom.canSelectTable()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Select Table operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the insertion point or selection is within a table.

### **dom.canSetLinkHref()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can change the link around the current selection or create one if necessary.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection is an image, text, or an insertion point inside a link. A text selection is defined as a selection for which the text Property inspector would appear.

### **dom.canShowListPropertiesDialog()**

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can show the List Properties dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection is within an LI tag.
<b>Enabler</b>	None.

### **dom.canSplitFrame()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Split Frame [Left   Right   Up   Down] operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection is within a frame.

### **dom.canSplitTableCell()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Split Cell operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the insertion point is inside a table cell or the selection is a table cell.

### **dom.canStopPlugin()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Stop operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection is currently being played with a plugin.

### **dom.canUndo()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Undo operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether any steps remain to undo.

### **dom.hasTracingImage()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether the document has a tracing image.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the document has a tracing image.

### **dreamweaver.canClipCopy()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Copy operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether there is anything selected that can be copied to the clipboard.

### **dreamweaver.canClipCut()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Cut operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether there is anything selected that can be cut to the clipboard.

### **dreamweaver.canClipPaste()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Paste operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the clipboard contains any content that can be pasted into the current document or the active pane in the Site window; or, on the Macintosh, an edit field in a floating palette or dialog box.

### **dreamweaver.canDeleteSelection()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can delete the current selection. Depending on the window that has focus, the deletion may occur in the Document window or the Site window; or, on the Macintosh, in an edit field in a dialog box or floating palette.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection is a range (that is, not an insertion point).

### **dreamweaver.canExportCSS()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Export CSS Styles operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the document contains any class styles defined in the HEAD.

### **dreamweaver.canFindNext()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Find Next operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether a search pattern has already been established.

### **dreamweaver.canOpenInFrame()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Open in Frame operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection or insertion point is within a frame.

### **dreamweaver.canPlayRecordedCommand()**

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a Play Recorded Command operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether there is an active document and a previously-recorded command that can be played.

### **dreamweaver.canRedo()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Redo operation in the current context.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether any operations can be undone.

### **dreamweaver.canRevertDocument()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Revert (to the last-saved version) operation.
<b>Arguments</b>	<i>documentObject</i>  The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code> ).
<b>Returns</b>	A Boolean value indicating whether the document is in an unsaved state and a saved version of the document exists on a local drive.

### **dreamweaver.canSaveAll()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Save All operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether one or more unsaved documents are open.

### **dreamweaver.canSaveDocument()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Save operation on the specified document.
<b>Arguments</b>	<i>documentObject</i>  The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code> ).
<b>Returns</b>	A Boolean value indicating whether the document has any unsaved changes.

### **dreamweaver.canSaveDocumentAsTemplate()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Save As Template operation on the specified document.
<b>Arguments</b>	<i>documentObject</i>  The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code> ).
<b>Returns</b>	A Boolean value indicating whether the document can be saved as a template.

### **`dreamweaver.canSaveFrameset()`**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Save Frameset operation on the specified document.
<b>Arguments</b>	<i>documentObject</i>  The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code> ).
<b>Returns</b>	A Boolean value indicating whether the document is a frameset with unsaved changes.

### **`dreamweaver.canSaveFramesetAs()`**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Save Frameset As operation on the specified document.
<b>Arguments</b>	<i>documentObject</i>  The argument is the object at the root of a document's DOM tree (the value returned by <code>dreamweaver.getDocumentDOM()</code> ).
<b>Returns</b>	A Boolean value indicating whether the document is a frameset.

### **`dreamweaver.canSelectAll()`**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Select All operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether a Select All operation can be performed.

### **`dreamweaver.canShowFindDialog()`**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Find operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the Site window or a Document window is open. This function returns <code>FALSE</code> when the selection is in the HEAD.



### **dreamweaver.canUndo()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Undo operation in the current context.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether any operations can be undone.

### **dreamweaver.isRecording()**

<b>Availability</b>	3.0
<b>Description</b>	Reports whether Dreamweaver is currently recording a command.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether Dreamweaver is recording a command.

### **dreamweaver.htmlStylePalette.canEditSelection()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can edit, delete, or duplicate the selection in the HTML Style palette.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value. This function returns FALSE if no style is selected, or if one of the “clear” styles is selected.

### **dreamweaver.timelineInspector.canAddFrame()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Add Frame operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the Timeline inspector has any animation bars or behaviors.

### **dreamweaver.timelineInspector.canAddKeyFrame()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Add Keyframe operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection in the Timeline inspector is part of an animation bar.

### **`dreamweaver.timelineInspector.canChangeObject()`**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Change Object operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection in the Timeline inspector is part of an animation bar.

### **`dreamweaver.timelineInspector.canRemoveBehavior()`**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Remove Behavior operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the selection in the Timeline inspector is a behavior.

### **`dreamweaver.timelineInspector.canRemoveFrame()`**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Remove Frame operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the Timeline inspector has any animation bars or behaviors.

### **`dreamweaver.timelineInspector.canRemoveKeyFrame()`**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Remove Keyframe operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the current frame in the Timeline inspector is a keyframe.

### **`dreamweaver.timelineInspector.canRemoveObject()`**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Remove Object operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the Timeline inspector has any animation bars.

### site.canAddLink()

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform an Add Link to [Existing File   New File] operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating that the selected document in the site map is an HTML file.

### site.canChangeLink()

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Change Link operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating that an HTML or Flash file links to the selected file in the site map.

### site.canCheckIn()

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a Check In operation.
<b>Arguments</b>	<i>siteOrURL</i>  The argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.
<b>Returns</b>	A Boolean value indicating whether all of the following conditions are true: <ul style="list-style-type: none"><li>• A remote site has been defined.</li><li>• If a Document window has focus, the file has been saved in a local site; or, if the Site window has focus, one or more files or folders is selected.</li><li>• Check in/check out is turned on.</li></ul>

### **site.canCheckOut()**

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a Check Out operation on the specified file or files.
<b>Arguments</b>	<i>siteOrURL</i>  The argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.
<b>Returns</b>	A Boolean value indicating whether all of the following conditions are true: <ul style="list-style-type: none"><li>• A remote site has been defined.</li><li>• If a Document window has focus, the file is part of a local site and is not already checked out; or, if the Site window has focus, one or more files or folders is selected and at least one of the selected files is not already checked out.</li><li>• Check in/check out is turned on.</li></ul>

### **site.canConnect()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can connect to the remote site.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the current remote site is an FTP site.

### **site.canFindLinkSource()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Find Link Source operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating that the selected link in the site map is not the home page.

### site.canGet()

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a Get operation.
<b>Arguments</b>	<i>siteOrURL</i>  The argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.
<b>Returns</b>	If the argument is "site", Boolean value indicating whether one or more files or folders is selected in the Site window and a remote site has been defined. If the argument is a URL, a Boolean value indicating whether the document belongs to a site for which a remote site has been defined.

### site.canLocateInSite()

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a Locate in Local Site or Locate in Remote Site operation (depending on the argument).
<b>Arguments</b>	<i>localOrRemote, siteOrURL</i> <ul style="list-style-type: none"><li>• The first argument must be either "local" or "remote".</li><li>• The second argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.</li></ul>
<b>Returns</b>	One of the following values: <ul style="list-style-type: none"><li>• If the second argument is "site", a Boolean value indicating whether both panes contain site files (not the site map) and whether the selection is in the opposite pane from the argument.</li><li>• If the first argument is "local" and the second argument is a URL, a Boolean value indicating whether the document belongs to a site.</li><li>• If the first argument is "remote" and the second argument is a URL, a Boolean value indicating whether the document belongs to a site for which a remote site has been defined, and, if the server type is Local/Network, whether the drive is mounted.</li></ul>

### site.canMakeEditable()

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Turn Off Read Only operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether one or more of the selected files is locked.

### **site.canMakeNewFileOrFolder()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a New File or New Folder operation in the Site window.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether any files are visible in the selected pane of the Site window.

### **site.canOpen()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can open the files or folders that are currently selected in the Site window.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether any files or folders are selected in the Site window.

### **site.canPut()**

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a Put operation.
<b>Arguments</b>	<i>siteOrURL</i>  The argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.
<b>Returns</b>	If the argument is "site", a Boolean value indicating whether any files or folders are selected in the Site window and a remote site has been defined. If the argument is a URL, a Boolean value indicating whether the document belongs to a site for which a remote site has been defined.

### **site.canRecreateCache()**

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Recreate Site Cache operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the Use Cache To Speed Link Updates option is enabled for the current site.

### site.canRefresh()

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Refresh [Local   Remote] operation.
<b>Arguments</b>	<i>localOrRemote</i>  The argument must be either "local" or "remote".
<b>Returns</b>	TRUE if <i>localOrRemote</i> is "local"; otherwise, a Boolean value indicating whether a remote site has been defined.

### site.canRemoveLink()

<b>Availability</b>	3.0
<b>Description</b>	Checks whether Dreamweaver can perform a Remove Link operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating that an HTML or Flash file links to the selected file in the site map.

### site.canSetLayout()

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a Layout operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the site map is visible.

### site.canSelectNewer()

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a Select Newer [Remote   Local] operation.
<b>Arguments</b>	<i>localOrRemote</i>  The argument must be either "local" or "remote".
<b>Returns</b>	A Boolean value whether the document belongs to a site for which a remote site has been defined.

### **site.canShowPageTitles()**

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a Show Page Titles operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the site map is visible.

### **site.canSynchronize()**

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a Synchronize operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether a remote site has been defined.

### **site.canUndoCheckout()**

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform an Undo Checkout operation.
<b>Arguments</b>	<i>siteOrURL</i>  The argument must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.
<b>Returns</b>	A Boolean value indicating whether the specified file or at least one of the selected files is checked out (by any user).

### **site.canViewAsRoot()**

<b>Availability</b>	3.0
<b>Description</b>	Determines whether Dreamweaver can perform a View as Root operation.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the specified file is an HTML or Flash file.



## CHAPTER 4

### The File I/O API

---

Dreamweaver 3 includes a C shared library called DWfile that gives authors of objects, commands, behaviors, data translators, floating palettes, and property inspectors the ability to read and write files on the local file system. This chapter describes the file I/O API and how to use it.

For general information on how C libraries interact with the JavaScript interpreter in Dreamweaver, see “C-Level Extensibility” on page 209.

### Verifying that DWfile is installed

To access the functions in the DWfile library, the library must be present in the Configuration/JSExtensions folder and loaded by Dreamweaver. Because DWfile did not ship with Dreamweaver 2 (it was available as a separate download that had to be installed by the user), it is useful to verify that the library is available before calling any of its functions. You can do this by checking the `typeof(DWfile)`. If DWfile does not exist, `typeof(DWfile)` will return `undefined`. For example, if you are using DWfile in the context of a command, you might check for the existence of DWfile as part of the `canAcceptCommand()` function:

```
// Returns TRUE if typeof(DWfile) is not undefined, FALSE otherwise.
function canAcceptCommand(){
    return (typeof(DWfile) != "undefined");
}
```

## The file I/O API

All of the functions in the file I/O API are methods of the DWfile object. Optional arguments are enclosed in curly braces ({ }). Functions added for Dreamweaver 3 have an availability of 3.0. Functions with an availability of 2.0 were included in the version of DWfile that was supplied as a download for Dreamweaver 2 from the Macromedia web site. This version of DWfile may have been installed with some third-party objects.

### DWfile.copy()

<b>Availability</b>	3.0
<b>Description</b>	Copies the specified file to the specified URL.
<b>Arguments</b>	<i>originalURL</i> , <i>copyURL</i> <ul style="list-style-type: none"><li>• The first argument is the file you want to copy, expressed as a file:// URL.</li><li>• The second argument is the location to which the copied file should be saved, expressed as a file:// URL.</li></ul>
<b>Returns</b>	TRUE if the copy succeeded, FALSE otherwise.
<b>Example</b>	The following code copies a file called myconfig.cfg to myconfig_backup.cfg. <pre>var fileURL = "file:///c:/Config/myconfig.cfg"; var newURL = "file:///c:/Config/myconfig_backup.cfg"; DWfile.copy(fileURL, newURL);</pre>

### DWfile.createFolder()

<b>Availability</b>	2.0
<b>Description</b>	Creates a folder (directory) at the specified location.
<b>Arguments</b>	<i>folderURL</i> <p>The argument is the location of the folder you want to create, expressed as a file:// URL.</p>
<b>Returns</b>	TRUE if the folder was successfully created, FALSE otherwise.
<b>Example</b>	The following code attempts to create a folder called tempFolder at the top level of the C drive and displays an alert box indicating whether the operation was successful. <pre>var folderURL = "file:///c:/tempFolder"; if (DWfile.createFolder(folderURL)){     alert("Created " + folderURL); }else{     alert("Unable to create " + folderURL); }</pre>

## DWfile.exists()

<b>Availability</b>	2.0
<b>Description</b>	Tests for the existence of the specified file.
<b>Arguments</b>	<i>fileURL</i>  The argument is the file you are looking for, expressed as a file:// URL.
<b>Returns</b>	TRUE if the file exists, FALSE otherwise.
<b>Example</b>	The following code checks for a file called mydata.txt and displays an alert box that tells the user whether the file exists or not.  <pre>var fileURL = "file:///c:/temp/mydata.txt"; if (DWfile.exists(fileURL)){     alert( fileURL + " exists!"); }else{     alert( fileURL + " does not exist."); }</pre>

## DWfile.getAttributes()

<b>Availability</b>	2.0
<b>Description</b>	Gets the attributes of the specified file or folder.
<b>Arguments</b>	<i>fileURL</i>  The argument is the file or folder for which you want to get attributes, expressed as a file:// URL.
<b>Returns</b>	A string representing the attributes of the specified file or folder, or NULL if the file or folder does not exist. Characters in the string represent the attributes as follows: <ul style="list-style-type: none"><li>• R is read only.</li><li>• D is folder (directory).</li><li>• H is hidden.</li><li>• S is system file or folder.</li></ul>
<b>Example</b>	The following code gets the attributes of the file mydata.txt and displays an alert box if the file is read only.  <pre>var URL = "file:///c:/temp/mydata.txt"; var str = DWfile.getAttributes(URL); if (str &amp;&amp; (str.indexOf("R") != -1)){     alert(URL + " is read only!"); }</pre>

## DWfile.getModificationDate()

**Availability** 2.0

**Description** Gets the time when the file was last modified.

**Arguments** *fileURL*

The argument is the file for which you are checking the last-modified time, expressed as a file:// URL.

**Returns** A string containing a hexadecimal number that represents the number of time units that have elapsed since some base time. The exact meaning of time units and base time is platform dependent; in Windows, for example, a time unit is 100ns, and the base time is January 1st, 1600.

**Example** It's most useful to call the function twice and compare the return values because the value returned by this function is platform-dependent and is not a recognizable date and time. For example, the following code gets the modification dates of file1.txt and file2.txt and displays an alert box indicating which file is newer.

```
var file1 = "file:///c:/temp/file1.txt";
var file2 = "file:///c:/temp/file2.txt";
var time1 = DWfile.getModificationDate(file1);
var time2 = DWfile.getModificationDate(file2);
if (time1 == time2){
    alert("file1 and file2 were saved at the same time");
}else if (time1 < time2){
    alert("file1 older than file2");
}else{
    alert("file1 is newer than file2");
}
```

## DWfile.listFolder()

<b>Availability</b>	2.0
<b>Description</b>	Gets a list of the contents of the specified folder.
<b>Arguments</b>	<i>folderURL</i> , { <i>constraint</i> }
	<ul style="list-style-type: none"><li>• The first argument is the folder for which you want a contents list, expressed as a file:// URL, plus an optional wildcard file mask. Valid wildcards are * (matches 1 or more characters) and ? (matches a single character).</li><li>• The second argument, if supplied, must be either "files" (return only files) or "directories" (return only directories). If omitted, the function returns both files and directories.</li></ul>
<b>Returns</b>	An array of strings representing the contents of the folder.
<b>Example</b>	The following code gets a list of all the text (.txt) files in the temp folder and displays the list in an alert box.

```
var folderURL = "file:///c:/temp";
var fileMask = "*.txt";
var list = listFolder(folderURL + "/" + fileMask, "files");
if (list){
    alert(folderURL + " contains: " + list.join("\n"));
}
```

## DWfile.read()

<b>Availability</b>	2.0
<b>Description</b>	Reads the contents of the specified file into a string.
<b>Arguments</b>	<i>fileURL</i>
	The argument is the file you want to read, expressed as a file:// URL.
<b>Returns</b>	A string containing the contents of the file, or NULL if the read fails.
<b>Example</b>	The following code reads the file mydata.txt and, if successful, displays an alert box with the contents of the file.

```
var fileURL = "file:///c:/temp/mydata.txt";
var str = DWfile.read( fileURL);
if (str){
    alert( fileURL + " contains: " + str);
}
```

## DWfile.remove()

<b>Availability</b>	3.0
<b>Description</b>	Moves the specified file to the Recycling Bin or Trash.  <b>Note:</b> Windows displays a confirmation dialog box if the file is read-only.
<b>Arguments</b>	<i>fileURL</i>  The argument is the file you want to remove, expressed as a file:// URL.
<b>Returns</b>	TRUE if the operation succeeded, FALSE otherwise.

## DWfile.write()

<b>Availability</b>	2.0
<b>Description</b>	Writes the specified string to the specified file. If the specified file does not yet exist, it is created.
<b>Arguments</b>	<i>fileURL, text, {mode}</i> <ul style="list-style-type: none"><li>• The first argument is the file you are writing to, expressed as a file:// URL.</li><li>• The second argument is the string to be written.</li><li>• The third argument, if supplied, must be "append". If this argument is omitted, the contents of the file are overwritten by the string.</li></ul>
<b>Returns</b>	TRUE if the string was successfully written to the file, FALSE otherwise.
<b>Example</b>	<p>The following code attempts to write the string "xxx" to the file mydata.txt and displays an alert if the write succeeded. It then attempts to append the string "aaa" to the file and displays a second alert if the write succeeded. After executing this script, the file mydata.txt will contain the text xxxaaa and nothing else.</p> <pre>var fileURL = "file:///c:/temp/mydata.txt"; if (DWfile.write(fileURL, "xxx")){     alert("Wrote xxx to " + fileURL); } if (DWfile.write(fileURL, "aaa", "append")){     alert("Appended aaa to " + fileURL); }</pre>

## CHAPTER 5

### The Design Notes API

---

Both Dreamweaver 3 and Fireworks 3 offer a new way for web designers and developers to store and retrieve extra information about documents—information such as review comments, change notes, or the source file for a GIF or JPEG—in files called Design Notes.

MMNotes is a C shared library that lets extensions authors read and write Design Notes files. Like the DWfile shared library, MMNotes has a JavaScript API that makes it possible to call the functions contained in the library from objects, comands, behaviors, floating palettes, property inspectors, and data translators.

What's different about MMNotes is that it also has a C API that gives other applications an opportunity to read and write Design Notes files. The MMNotes shared library can be used independently of Dreamweaver, even if Dreamweaver is not installed.

For more information about using the Design Notes feature from within Dreamweaver, see *Using Dreamweaver*.

## How Design Notes work

Each Design Notes file stores information for a single document. If one or more documents in a directory has a Design Notes file associated with it, Dreamweaver creates a `_notes` subfolder in that directory in which to store Design Notes files. The `_notes` folder and the Design Notes files it contains are not visible in the Site window, but they show up in the Finder or Windows Explorer. A Design Notes file name is made up of the main file name plus the extension `.mno`. For example, the Design Notes file associated with `avocado8.gif` is `avocado8.gif.mno`.

Design Notes files are XML files that store information in a series of key/value pairs. The key describes the type of information that is being stored, and the value represents the information itself. Keys are limited to 64 characters.

The Design Notes file for `foghorn.gif` might look like this:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<info>
  <infoitem key="FW_source" value="file:///C:/sites/dreamcentral/images/sourceFiles/
foghorn.png" />
  <infoitem key="Author" value="Heidi B." />
  <infoitem key="Status" value="Final draft, approved by Jay L." />
</info>
```

## The Design Notes JavaScript API

All of the functions in the Design Notes JavaScript API are methods of the `MMNotes` object. Optional arguments are enclosed in curly braces (`{ }`).

### `MMNotes.open()`

<b>Description</b>	Opens the Design Notes file associated with the specified file, or creates one if none yet exists.
<b>Arguments</b>	<i>filePath</i> , { <i>bForceCreate</i> }
	<ul style="list-style-type: none"><li>• The first argument is the path to the main file with which the Design Notes file is associated, expressed as a <code>file://</code> URL.</li><li>• The second argument is a Boolean value indicating whether to create the Note even if Design Notes are turned off for the site or <i>filePath</i> is not associated with any site.</li></ul>
<b>Returns</b>	The file handle for the Design Notes file, or zero (0) if the file was not opened or created.
<b>Example</b>	See “ <code>MMNotes.set()</code> ” on page 185.



### MMNotes.close()

<b>Description</b>	Closes the specified Design Notes file and saves any changes. If all key/value pairs have been removed, Dreamweaver deletes the Design Notes file.
<b>Arguments</b>	<i>fileHandle</i>  The argument is the file handle returned by MMNotes.open().
<b>Returns</b>	Nothing.
<b>Example</b>	See “MMNotes.set()” on page 185.

### MMNotes.set()

<b>Description</b>	Creates or updates one key/value pair in a Design Notes file.
<b>Arguments</b>	<i>fileHandle, keyName, valueString</i> <ul style="list-style-type: none"><li>• The first argument is the file handle returned by MMNotes.open().</li><li>• The second argument is a string containing the name of the key.</li><li>• The third argument is a string containing the value.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful.
<b>Example</b>	The following code opens the Design Notes file associated with a file in the dreamcentral site called peakhike99/index.html, adds a new key/value pair, changes the value of an existing key, and then closes the Note file.  <pre>var noteHandle = MMNotes.open('file:///c:/sites/dreamcentral/ peakhike99/index.html',TRUE); MMNotes.set(noteHandle,"Author","M. G. Miller"); MMNotes.set(noteHandle,"Last Changed","August 28, 1999"); MMNotes.close(noteHandle);</pre>

### MMNotes.remove()

<b>Description</b>	Removes the specified key (and its value) from the specified Design Notes file.
<b>Arguments</b>	<i>fileHandle, keyName</i> <ul style="list-style-type: none"><li>• The first argument is the file handle returned by MMNotes.open().</li><li>• The second argument is a string containing the name of the key to be removed.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful.

### MMNotes.get()

<b>Description</b>	Gets the value of the specified key in the specified Design Notes file.
<b>Arguments</b>	<i>fileHandle</i> , <i>keyName</i> <ul style="list-style-type: none"><li>• The first argument is the file handle returned by MMNotes.open().</li><li>• The second argument is a string containing the name of the key.</li></ul>
<b>Returns</b>	A string containing the value of the key.
<b>Example</b>	See “MMNotes.getKeys()” on page 186.

### MMNotes.getKeyCount()

<b>Description</b>	Gets the number of key/value pairs in the specified Design Notes file.
<b>Arguments</b>	<i>fileHandle</i> <p>The argument is the file handle returned by MMNotes.open().</p>
<b>Returns</b>	An integer representing the number of key/value pairs in the Design Notes file.

### MMNotes.getKeys()

<b>Description</b>	Gets a list of all the keys in a Design Notes file.
<b>Arguments</b>	<i>fileHandle</i> <p>The argument is the file handle returned by MMNotes.open().</p>
<b>Returns</b>	Array of strings, each containing the name of a key.
<b>Example</b>	<p>The following code might be used in a custom floating palette to display the Design Notes information for the active document:</p> <pre>var noteHandle = MMNotes.open(dw.getDocumentDOM().URL); var theKeys = MMNotes.getKeys(noteHandle); var noteString = ""; var theValue = ""; for (var i=0; i &lt; theKeys.length; i++){     theValue = MMNotes.get(noteHandle,theKeys[i]);     noteString += theKeys[i] + " = " theValue + "\n"; } document.theForm.bigTextField.value = noteString;</pre>

### MMNotes.getSiteRootForFile()

<b>Description</b>	Determines the site root for the specified Design Notes file.
<b>Arguments</b>	<i>fileURL</i>  The argument is the path to a local file, expressed as a file:// URL.
<b>Returns</b>	A string containing the path of the Local Root Folder for the site, expressed as a file:// URL, or an empty string if Dreamweaver is not installed or the Design Notes file is outside of any site.

### MMNotes.getVersionNum()

<b>Description</b>	Gets the version number of the MMNotes shared library.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the version number.

### MMNotes.getVersionName()

<b>Description</b>	Gets the version name of the MMNotes shared library, which indicates the application that implemented it.
<b>Arguments</b>	None.
<b>Returns</b>	A string containing the name of the application that implemented MMNotes shared library.
<b>Example</b>	Calling MMNotes.getVersionName() from a Dreamweaver command, object, behavior, property inspector, floating palette, or data translator returns "Dreamweaver". Calling MMNotes.getVersionName() from Fireworks also returns "Dreamweaver" because Fireworks uses the same version of the library (the one created by the Dreamweaver engineering team).

### MMNotes.filePathToLocalURL()

<b>Description</b>	Converts the specified local drive path to a file:// URL.
<b>Arguments</b>	<i>drivePath</i>  The argument is a string containing the full drive path.
<b>Returns</b>	A string containing the file:// URL for the specified file.
<b>Example</b>	A call to MMNotes.filePathToLocalURL('C:/sites/webdev/index.htm') returns "file:///c:/sites/webdev/index.htm".

### MMNotes.localURLToFilePath()

<b>Description</b>	Converts the specified file:// URL to a local drive path.
<b>Arguments</b>	<i>fileURL</i>  The argument is the path to a local file, expressed as a file:// URL.
<b>Returns</b>	A string containing the local drive path for the specified file.
<b>Example</b>	A call to MMNotes.localURLToFilePath("file:///MacintoshHD/images/moon.gif") returns "MacintoshHD:images:moon.gif".

## The Design Notes C API

In addition to the JavaScript API, the MMNotes shared library also exposes a C API that lets other applications create Design Notes files. It is not necessary to call these C functions directly if you are using the MMNotes shared library in Dreamweaver; the JavaScript versions of the functions call them for you.

This section contains descriptions of the functions, their arguments, and their return values; definitions for all of the functions and data types are in the MMInfo.h file in the Extending/c\_files folder inside the Dreamweaver application folder.

Optional arguments are enclosed in curly braces ({ }).

### FileHandle OpenNotesFile()

<b>Description</b>	Opens the Design Notes file associated with the specified file, or creates one if none yet exists.
<b>Arguments</b>	<code>const char* localFileURL, {BOOL bForceCreate}</code> <ul style="list-style-type: none"><li>• The first argument is a string containing the path to the main file with which the Design Notes file is associated, expressed as a file:// URL.</li><li>• The second argument is a Boolean value indicating whether to create the Design Notes file even if Design Notes are turned off for the site or <i>filePath</i> is not associated with any site.</li></ul>

### void CloseNotesFile()

<b>Description</b>	Closes the specified Design Notes file and saves any changes. If all key/value pairs have been removed from the Note, Dreamweaver deletes it.
<b>Arguments</b>	<i>FileHandle noteHandle</i>  The argument is the file handle returned by OpenNotesFile().
<b>Returns</b>	Nothing.

### BOOL SetNote()

<b>Description</b>	Creates or updates one key/value pair in a Design Notes file.
<b>Arguments</b>	<i>FileHandle noteHandle</i> , <i>const char keyName[64]</i> , <i>const char* value</i> <ul style="list-style-type: none"><li>• The first argument is the file handle returned by <code>OpenNotesFile()</code>.</li><li>• The second argument is a string containing the name of the key.</li><li>• The third argument is a string containing the value.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful.

### BOOL RemoveNote()

<b>Description</b>	Removes the specified key (and its value) from the specified Design Notes file.
<b>Arguments</b>	<i>FileHandle noteHandle</i> , <i>const char keyName[64]</i> <ul style="list-style-type: none"><li>• The first argument is the file handle returned by <code>OpenNotesFile()</code>.</li><li>• The second argument is a string containing the name of the key to be removed.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful.

### int GetNoteLength()

<b>Description</b>	Gets the length of the value associated with the specified key.
<b>Arguments</b>	<i>FileHandle noteHandle</i> , <i>const char keyName[64]</i> <ul style="list-style-type: none"><li>• The first argument is the file handle returned by <code>OpenNotesFile()</code>.</li><li>• The second argument is a string containing the name of the key.</li></ul>
<b>Returns</b>	An integer representing the length of the value.
<b>Example</b>	See “BOOL GetNote()” on page 190.

## BOOL GetNote()

<b>Description</b>	Gets the value of the specified key in the specified Design Notes file.
<b>Arguments</b>	<p>FileHandle <i>noteHandle</i>, const char <i>keyName</i>[64], char* <i>valueBuf</i>, int <i>valueBufLength</i></p> <ul style="list-style-type: none"><li>• The first argument is the file handle returned by <code>OpenNotesFile()</code>.</li><li>• The second argument is a string containing the name of the key.</li><li>• The third argument is the buffer where the value should be stored.</li><li>• The fourth argument is the integer returned by <code>GetNoteLength(<i>noteHandle</i>,<i>keyName</i>)</code>, indicating the maximum length of the value buffer.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful, and stores the value of the key in <i>valueBuf</i> .
<b>Example</b>	<p>The following code gets the value of the comments key in the Design Notes file associated with <code>welcome.html</code>:</p> <pre>FileHandle noteHandle = OpenNotesFile("file:///c /sites/avocado8/iwjs/welcome.html"); int valueLength = GetNotesLength( noteHandle, "comments"); char* valueBuffer = new char[valueLength + 1] GetNote(noteHandle, "comments", valueBuffer, valueLength + 1); printf("Comments: %s",valueBuffer); CloseNotesFile(noteHandle);</pre>

## int GetNotesKeyCount()

<b>Description</b>	Gets the number of key/value pairs in the specified Design Notes file.
<b>Arguments</b>	<p>FileHandle <i>noteHandle</i></p> <p>The argument is the file handle returned by <code>OpenNotesFile()</code>.</p>
<b>Returns</b>	An integer representing the number of key/value pairs in the Design Notes file.

## BOOL GetNotesKeys()

<b>Description</b>	Gets a list of all the keys in a Design Notes file.
<b>Arguments</b>	<p>FileHandle <i>noteHandle</i>, char* <i>keyBufArray</i>[64], int <i>keyArrayMaxLen</i></p> <ul style="list-style-type: none"><li>• The first argument is the file handle returned by <code>OpenNotesFile()</code>.</li><li>• The second argument is the buffer array where the keys should be stored.</li><li>• The third argument is the integer returned by <code>GetNotesKeyCount(<i>noteHandle</i>)</code>, indicating the maximum number of items in the key buffer array.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful, and stores the key names in <i>keyBufArray</i> .
<b>Example</b>	The following code prints the key names and values of all the keys in the Design Notes file associated with <code>welcome.html</code> :

```
typedef char[64] InfoKey;
FileHandle noteHandle = OpenNotesFile("file:///c|/sites/avocado8/iwjs/welcome.html");
if (noteHandle > 0){
    int keyCount = GetNotesKeyCount(noteHandle);
    if (keyCount <= 0)
        return;
    InfoKey* keys = new InfoKey[keyCount];
    BOOL succeeded = GetNotesKeys(noteHandle, keys, keyCount);

    if (succeeded){
        for (int i=0; i < keyCount; i++){
            printf("Key is: %s\n", keys[i]);
            printf("Value is: %s\n\n", GetNote(noteHandle, keys[i]));
        }
        delete keys;
    }
    CloseNotesFile(noteHandle);
}
```

## BOOL GetSiteRootForFile()

<b>Description</b>	Determines the site root for the specified Design Notes file.
<b>Arguments</b>	<p>char* <i>filePath</i>, char* <i>siteRootBuf</i>, int <i>siteRootBufMaxLen</i>, {InfoPrefs* <i>infoPrefs</i>}</p> <ul style="list-style-type: none"><li>• The first argument is the file handle returned by <code>OpenNotesFile()</code>.</li><li>• The second argument is the buffer where the site root should be stored.</li><li>• The third argument is the maximum size of <i>siteRootBuf</i>.</li><li>• The fourth argument is a reference to a struct in which the preferences for the site should be stored.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful, and stores the site root in <i>siteRootBuf</i> . If <i>infoPrefs</i> is specified, the function also returns the Design Notes preferences for the site. The InfoPrefs struct has two variables: <code>bUseDesignNotes</code> and <code>bUploadDesignNotes</code> , both of type BOOL.

### BOOL GetVersionNum()

<b>Description</b>	Gets the version number of the MMNotes shared library.
<b>Arguments</b>	<code>char* versionNumBuf</code> , <code>int versionNumBufMaxLen</code> <ul style="list-style-type: none"><li>• The first argument is the buffer where the version number should be stored.</li><li>• The second argument is the maximum size of <code>versionNumBuf</code>.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful, and stores the version number in <code>versionNumBuf</code> .

### BOOL GetVersionName()

<b>Description</b>	Gets the version name of the MMNotes shared library, which indicates the application that implemented it.
<b>Arguments</b>	<code>char* versionNameBuf</code> , <code>int versionNameBufMaxLen</code> <ul style="list-style-type: none"><li>• The first argument is the buffer where the version name should be stored.</li><li>• The second argument is the maximum size of <code>versionNameBuf</code>.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful, and stores the version name in <code>versionNameBuf</code> .

### BOOL FilePathToLocalURL()

<b>Description</b>	Converts the specified local drive path to a file:// URL.
<b>Arguments</b>	<code>const char* drivePath</code> , <code>char* localURLBuf</code> , <code>int localURLMaxLen</code> <ul style="list-style-type: none"><li>• The first argument is a string containing the full drive path.</li><li>• The second argument is the buffer where the file:// URL should be stored..</li><li>• The third argument is the maximum size of <code>localURLBuf</code>.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful, and stores the file:// URL in <code>localURLBuf</code> .

### BOOL LocalURLToFilePath()

<b>Description</b>	Converts the specified file:// URL to a local drive path.
<b>Arguments</b>	<code>const char* localURL</code> , <code>char* drivePathBuf</code> , <code>int drivePathMaxLen</code> <ul style="list-style-type: none"><li>• The first argument is the path to a local file, expressed as a file:// URL.</li><li>• The second argument is the buffer where the local drive path should be stored.</li><li>• The third argument is the maximum size of <code>drivePathBuf</code>.</li></ul>
<b>Returns</b>	A Boolean value indicating whether the operation was successful, and stores the local drive path in <code>drivePathBuf</code> .



## CHAPTER 6

### The Fireworks Integration API

---

FWLaunch is a C shared library that gives authors of objects, commands, behaviors, and property inspectors the ability to communicate with Fireworks. This chapter describes the Fireworks integration API and how to use it; for general information on how C libraries interact with the JavaScript interpreter in Dreamweaver, see “C-Level Extensibility” on page 209

#### The Fireworks integration API

All functions in the Fireworks integration API are methods of the FWLaunch object. Optional arguments are enclosed in curly braces ({ }).

##### **FWLaunch.bringDWToFront()**

<b>Availability</b>	Dreamweaver 3.0, Fireworks 3.0
<b>Description</b>	Brings Dreamweaver to the front.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.

##### **FWLaunch.bringFWToFront()**

<b>Availability</b>	Dreamweaver 3.0, Fireworks 3.0
<b>Description</b>	Brings Fireworks to the front if it is running.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.

## FWLaunch.execJsInFireworks()

<b>Availability</b>	Dreamweaver 3.0, Fireworks 3.0
<b>Description</b>	Passes the specified string of JavaScript to Fireworks for execution.
<b>Arguments</b>	<p><i>javascriptOrFileURL</i></p> <p>The argument is either a string of literal JavaScript or the path to a .js or .jsf file, expressed as a file:// URL.</p>
<b>Returns</b>	<p>A cookie object if the JavaScript was passed successfully, or a nonzero error code indicating that one of the following errors has occurred:</p> <ul style="list-style-type: none"><li>• 1: Invalid usage; <i>javascriptOrFileURL</i> was specified as NULL or empty string, or the path to the .js or .jsf file was invalid.</li><li>• 2: File I/O error; Fireworks is unable to create a response file because the disk is full.</li><li>• 3: Error notifying Dreamweaver; the user is not running a valid version of Dreamweaver (3.0 or later).</li><li>• 4: Error launching Fireworks process; the function did not launch a valid version of Fireworks (3.0 or later).</li><li>• 5: User canceled the operation.</li></ul>

## FWLaunch.getJsResponse()

<b>Availability</b>	Dreamweaver 3.0, Fireworks 3.0
<b>Description</b>	Determines whether Fireworks is still executing the JavaScript passed to it by FWLaunch.execJsInFireworks(), whether the script has completed successfully, or whether an error has occurred.
<b>Arguments</b>	<p><i>progressTrackerCookie</i></p> <p>The argument is the cookie object returned by FWLaunch.execJsInFireworks().</p>

**Returns** A string containing the result of the script passed to `FWLaunch.execJsInFireworks()` if the operation completed successfully, `NULL` if Fireworks is still executing the JavaScript, or a nonzero error code indicating that one of the following errors has occurred:

- 1: Invalid usage; a JavaScript error occurred as Fireworks was executing the script.
- 2: File I/O error; Fireworks is unable to create a response file because the disk is full.
- 3: Error notifying Dreamweaver; the user is not running a valid version of Dreamweaver (3.0 or later).
- 4: Error launching Fireworks process; the function did not launch a valid version of Fireworks (3.0 or later).
- 5: User canceled the operation.

**Returns** The following code passes the string `"prompt('Please enter your name:')" to FWLaunch.execJsInFireworks() and then checks for the result:`

```
var progressCookie = FWLaunch.execJsInFireworks("prompt('Please enter your name:');");
var doneFlag = false;
while (!doneFlag){
    // check for completion every 1/2 second
    setTimeout('checkForCompletion()',500);
}

function checkForCompletion(){
    if (progressCookie != null) {
        var response = FWLaunch.getJsResponse(progressCookie);
        if (response != null) {
            if (typeof(response) == "number") {
                // error or user-cancel, time to close the window and let the user know we got an error
                window.close();
                alert("An error occurred.");
            }else{
                // got a valid response!
                alert("Nice to meet you, " + response);
                window.close();
            }
        }
        doneFlag = true;
    }
}
```

## FWLaunch.mayLaunchFireworks()

<b>Availability</b>	Dreamweaver 2.0, Fireworks 2.0
<b>Description</b>	Determines whether it is possible to launch a Fireworks optimization session.
<b>Arguments</b>	None.
<b>Returns</b>	A Boolean value indicating whether the platform is Windows, or if Macintosh, whether another Fireworks optimization session is not already running.

## FWLaunch.optimizeInFireworks()

<b>Availability</b>	Dreamweaver 2.0, Fireworks 2.0
<b>Description</b>	Launches a Fireworks optimization session for the specified image.
<b>Arguments</b>	<i>docURL</i> , <i>imageURL</i> , { <i>targetWidth</i> }, { <i>targetHeight</i> } <ul style="list-style-type: none"><li>• The first argument is the path to the active document, expressed as a file:// URL.</li><li>• The second argument is the path to the selected image. If the path is relative, it is relative to <i>docURL</i>.</li><li>• The third argument, if supplied, is the width to which the image should be resized.</li><li>• The fourth argument, if supplied, is the height to which the image should be resized.</li></ul>
<b>Returns</b>	0 if a Fireworks optimization session is successfully launched for the specified image; otherwise, a nonzero error code indicating that one of the following errors has occurred: <ul style="list-style-type: none"><li>• 1: Invalid usage; <i>docURL</i>, <i>imageURL</i>, or both were specified as NULL or empty string.</li><li>• 2: File I/O error; Fireworks is unable to create a response file because the disk is full.</li><li>• 3: Error notifying Dreamweaver; the user is not running a valid version of Dreamweaver (2.0 or later).</li><li>• 4: Error launching Fireworks process; the function did not launch a valid version of Fireworks (2.0 or later).</li><li>• 5: User canceled the operation.</li></ul>

## FWLaunch.validateFireworks()

<b>Availability</b>	Dreamweaver 2.0, Fireworks 2.0
<b>Description</b>	Looks for the specified version of Fireworks on the user's hard drive.
<b>Arguments</b>	<p><i>{versionNumber}</i></p> <p>The argument is a floating point number greater than or equal to 2.0; it represents the version of Fireworks that should be searched for. If this argument is omitted, the default is 2.0.</p>
<b>Returns</b>	A Boolean value indicating whether the specified version of Fireworks was found.
<b>Example</b>	<p>The following code checks whether Fireworks 3.0 is installed:</p> <pre>if (FWLaunch.validateFireworks(3.0)){     alert( "Fireworks 3.0 is installed."); }else{     alert( "Fireworks 3.0 is not installed."); }</pre>

## A simple Fireworks integration example

The following command asks Fireworks to prompt the user for her name, and then returns the name to Dreamweaver.

```
<html>
<head>
<title>Prompt in Fireworks</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<script>

function commandButtons(){
    return new Array("Prompt", "promptInFireworks()", "Cancel", "readyToCancel()",
"Close", "window.close()");
}

var gCancelClicked = false;
var gProgressTrackerCookie = null;

function readyToCancel() {
    gCancelClicked = true;
}

function promptInFireworks() {
    var isFireworks3 = FWLaunch.validateFireworks(3.0);
    if (!isFireworks3) {
        alert("You must have Fireworks 3.0 or later to use this command");
        return;
    }

    // Tell Fireworks to execute the prompt() method.
    gProgressTrackerCookie = FWLaunch.execJsInFireworks("prompt('Please enter your
name:')");

    // null means it wasn't launched, a number means an error code
    if (gProgressTrackerCookie == null || typeof(gProgressTrackerCookie) == "number") {
        window.close();
        alert("an error occurred");
        gProgressTrackerCookie = null;
    } else {
        // bring Fireworks to the front
        FWLaunch.bringFWToFront();
        // start the checking to see if Fireworks is done yet
        checkOneMoreTime();
    }
}

function checkOneMoreTime() {
    // Call checkJsResponse() every 1/2 second to see if Fireworks is done yet
    window.setTimeout("checkJsResponse()", 500);
}
```

```

function checkJsResponse() {
    var response = null;

    // The user clicked the cancel button, close the window
    if (gCancelClicked) {
        window.close();
        alert("cancel clicked");
    } else {
        // We're still going, ask Fireworks how it's doing
        if (gProgressTrackerCookie != null)
            response = FWLaunch.getJsResponse(gProgressTrackerCookie);

        if (response == null) {
            // still waiting for a response, call us again in 1/2 a second
            checkOneMoreTime();
        } else if (typeof(response) == "number") {
            // if the response was a number, it means an error occurred
            // the user cancelled in Fireworks
            window.close();
            alert("an error occurred.");
        } else {
            // got a valid response! This return value might not always be a
            // useful one, since not all functions in Fireworks return a string,
            // but we know this one does, so we can show the user what we got.
            window.close();
            FWLaunch.bringDWToFront(); // bring Dreamweaver to the front
            alert("Nice to meet you, " + response + "!");
        }
    }
}

</script>
</head>
<body>
<form>
<table width="313" nowrap>
<tr>
<td>This command asks Fireworks to execute the prompt() function.
When you click Prompt, Fireworks comes forward and asks you to
enter a value into a dialog box. That value is then returned to
Dreamweaver and displayed in an alert.</td>
</tr>
</table>
</form>
</body>
</html>

```





## CHAPTER 7

### The HTTP API

---

In Dreamweaver 3, extensions are no longer limited to working within the local file system. Dreamweaver now provides a mechanism for getting information from and sending information to a web server via hypertext transfer protocol (HTTP). This chapter describes the HTTP API and how to use it.

## The HTTP API

All of the functions in the HTTP API are methods of the MMHttp object. Most take at least a URL as an argument, and most return an object. The default port for URL arguments is 80; to specify a port other than 80, append a colon and the port number to the URL. For example:

```
MMHttp.getText("http://www.myserver.com:8025");
```

For functions that return an object, the object has two properties: `statusCode` and `data`.

`statusCode` indicates the status of the operation; possible values include, but are not limited to:

- 200: Status OK
- 400: Unintelligible request
- 404: Requested URL not found
- 405: Server does not support requested method
- 500: Unknown server error
- 503: Server capacity reached

For a comprehensive list of status codes for your server, check with your Internet service provider or system administrator.

The value of the `data` property varies according to the function; possible values are specified in the individual function listings.

Functions that return an object also have a “callback” version. Callback functions allow other functions to execute while the web server processes an HTTP request. This is useful if you are making multiple HTTP requests from Dreamweaver. The callback version of a function passes its ID and return value directly to the function specified as its first argument.

Optional arguments are enclosed in curly braces (`{ }`).

## MMHttp.clearTemp()

<b>Description</b>	Deletes all files in the Configuration/Temp folder inside the Dreamweaver application folder.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Example</b>	<p>The following code, when saved in a file inside the Configuration/Shutdown folder, removes all of the files from the Configuration/Temp folder when the user quits Dreamweaver:</p> <pre>&lt;html&gt; &lt;head&gt; &lt;title&gt;Clean Up Temp Files on Shutdown&lt;/title&gt; &lt;/head&gt; &lt;body onLoad="MMHttp.clearTemp()"&gt; &lt;/body&gt; &lt;/html&gt;</pre>

## MMHttp.getFile()

<b>Description</b>	<p>Gets the file at the specified URL and saves it in the Configuration/Temp folder inside the Dreamweaver application folder on the user's hard drive. Dreamweaver automatically creates subfolders that mimic the directory structure of the server; for example, if the specified file is at <code>http://www.dreamcentral.com/people/index.html</code>, Dreamweaver stores the <code>index.html</code> file in the <code>people</code> folder inside the <code>www.dreamcentral.com</code> folder.</p>
<b>Arguments</b>	<p><i>URL</i>, {<i>prompt</i>}, {<i>saveURL</i>}, {<i>titleBarLabel</i>}</p> <ul style="list-style-type: none"><li>• The first argument is an absolute URL on a web server; if the “<code>http://</code>” part of the URL is omitted, it is assumed.</li><li>• The second argument is a Boolean value that specifies whether to prompt the user to save the file. If <i>saveURL</i> is outside the Configuration/Temp folder, a <i>prompt</i> value of <code>FALSE</code> is ignored for security reasons.</li><li>• The third argument is the location on the user's hard drive where the file should be saved, expressed as a <code>file://</code> URL. If <i>prompt</i> is <code>TRUE</code> or <i>saveURL</i> is outside the Configuration/Temp folder, the user can override <i>saveURL</i> in the Save dialog box.</li><li>• The fourth argument is the label that should appear in the title bar of the Save dialog box.</li></ul>

**Returns** An object that represents the reply from the server. The data property of this object is a string containing the location at which the file was saved, expressed as a file:// URL. Normally the statusCode property of the object contains the status code received from the server. However, if a disk error occurs while saving the file on the local drive, the statusCode property contains an integer representing one of the following error codes if the operation was not successful:

- 1: Unspecified error
- 2: File not found
- 3: Invalid path
- 4: Number of open files limit reached
- 5: Access denied
- 6: Invalid file handle
- 7: Cannot remove current working directory
- 8: No more directory entries
- 9: Error setting file pointer
- 10 Hardware error
- 11: Sharing violation
- 12: Lock violation
- 13: Disk full
- 14: End of file reached

**Example** The following code gets an HTML file, saves all the files in the Configuration/Temp folder, and then opens the local copy of the HTML file in a browser:

```
var httpReply = MMHttp.getFile("http://www.dreamcentral.com/people/profiles/scott.html",
false);
if (httpReply.statusCode == 200){
    var saveLoc = httpReply.data;
    dw.browseDocument(saveLoc);
}
```

## MMHttp.getFileCallback()

<b>Description</b>	<p>Gets the file at the specified URL, saves it in the Configuration/Temp folder inside the Dreamweaver application folder on the user's hard drive, and then calls the specified function with the request ID and reply result. When saving the file locally, Dreamweaver automatically creates subfolders that mimic the directory structure of the server; for example, if the specified file is at <code>http://www.dreamcentral.com/people/index.html</code>, Dreamweaver stores the <code>index.html</code> file in the <code>people</code> folder inside the <code>www.dreamcentral.com</code> folder.</p>
<b>Arguments</b>	<p><i>callbackFunction</i>, <i>URL</i>, <i>{prompt}</i>, <i>{saveURL}</i>, <i>{titleBarLabel}</i></p> <ul style="list-style-type: none"><li>• The first argument is the name of the JavaScript function to call when the HTTP request is complete.</li><li>• The second argument is an absolute URL on a web server; if the “<code>http://</code>” part of the URL is omitted, it is assumed.</li><li>• The third argument is a Boolean value that specifies whether to prompt the user to save the file. If <i>saveURL</i> is outside the Configuration/Temp folder, a <i>prompt</i> value of <code>FALSE</code> is ignored for security reasons.</li><li>• The fourth argument is the location on the user's hard drive where the file should be saved, expressed as a <code>file://</code> URL. If <i>prompt</i> is <code>TRUE</code> or <i>saveURL</i> is outside the Configuration/Temp folder, the user can override <i>saveURL</i> in the Save dialog box.</li><li>• The fifth argument is the label that should appear in the title bar of the Save dialog box.</li></ul>
<b>Returns</b>	<p>An object that represents the reply from the server. The <code>data</code> property of this object is a string containing the location at which the file was saved, expressed as a <code>file://</code> URL. Normally the <code>statusCode</code> property of the object contains the status code received from the server. However, if a disk error occurs while saving the file on the local drive, the <code>statusCode</code> property contains an integer representing an error code. See “<code>MMHttp.getFile()</code>” on page 203 for a list of possible error codes.</p>

## MMHttp.getText()

<b>Description</b>	Retrieves the content of the document at the specified URL.
<b>Arguments</b>	<i>URL</i>  The argument is an absolute URL on a web server; if the “http://” part of the URL is omitted, it is assumed.
<b>Returns</b>	An object that represents the reply from the server. The data property of this object is a string containing the contents of the document.
<b>Example</b>	The following code gets the content of a file on a web server and puts it in a new, untitled Dreamweaver document:  <pre>var httpReply = MMHttp.getText("http://www.dreamcentral.com/people/profiles/lori.html"); if (httpReply.statusCode == 200){     var newDoc = dw.createDocument();     newDoc.documentElement.outerHTML = httpReply.data; }</pre>

## MMHttp.getTextCallback()

<b>Description</b>	Retrieves the content of the document at the specified URL and passes it to the specified function.
<b>Arguments</b>	<i>callbackFunc</i> , <i>URL</i> <ul style="list-style-type: none"><li>• The first argument is the name of the JavaScript function to call when the HTTP request is complete.</li><li>• The second argument is an absolute URL on a web server; if the “http://” part of the URL is omitted, it is assumed.</li></ul>
<b>Returns</b>	An object that represents the reply from the server. The data property of this object is a string containing the contents of the document.
<b>Example</b>	The following code populates a form field with the text returned by the MMHttp.GetTextCallback() function or shows an error message if the function returned an error:  <pre>var requestID = MMHttp.getTextCallback("httpCallback", "www.dreamcentral.com/index.html")  function httpCallback(requestID,reply) {     if (reply.statusCode == 200) {         document.theForm.docContents.value = reply.data;     }else{         alert("Request #: " + requestID + "returned the following error: " + reply.statusCode);     } }</pre>

## MMHttp.postText()

<b>Description</b>	Performs an HTTP post of the specified data to the specified URL. Typically the data associated with a post operation is form-encoded text, but it could be any type of data that the server expects to receive.
<b>Arguments</b>	<i>URL, dataToPost, {contentType}</i> <ul style="list-style-type: none"><li>• The first argument is an absolute URL on a web server; if the “http://” part of the URL is omitted, it is assumed.</li><li>• The second argument is the data to be posted. If the third argument is “application/x-www-form-urlencoded” or omitted, <i>dataToPost</i> must be form encoded according to section 8.2.1 of the RFC 1866 specification (available at <a href="http://www.faqs.org/rfcs/rfc1866.html">http://www.faqs.org/rfcs/rfc1866.html</a>).</li><li>• The third argument is the content type of the data to be posted. If omitted, this argument defaults to “application/x-www-form-urlencoded”.</li></ul>
<b>Returns</b>	An object that represents the reply from the server. The data property of this object is a string containing the data resulting from the post operation.

## MMHttp.postTextCallback()

<b>Description</b>	Performs an HTTP post of the text to the specified URL and passes the reply from the server to the specified function. Typically the data associated with a post operation is form-encoded text, but it could be any type of data that the server expects to receive.
<b>Arguments</b>	<i>callbackFunc, URL, dataToPost, {contentType}</i> <ul style="list-style-type: none"><li>• The first argument is the name of the JavaScript function to call when the HTTP request is complete.</li><li>• The second argument is an absolute URL on a web server; if the “http://” part of the URL is omitted, it is assumed.</li><li>• The third argument is the data to be posted. If the third argument is “application/x-www-form-urlencoded” or omitted, <i>data</i> must be form encoded according to section 8.2.1 of the RFC 1866 specification (available at <a href="http://www.faqs.org/rfcs/rfc1866.html">http://www.faqs.org/rfcs/rfc1866.html</a>).</li><li>• The fourth argument is the content type of the data to be posted. If omitted, this argument defaults to “application/x-www-form-urlencoded”.</li></ul>
<b>Returns</b>	An object that represents the reply from the server. The data property of this object is a string containing the data resulting from the post operation.





## CHAPTER 8

### C-Level Extensibility

---

The C-level extensibility mechanism lets you implement Dreamweaver extensibility files using a combination of JavaScript and your own C code. You define functions using C, bundle them in a DLL or shared library, save the library in the Configuration/JSExtensions folder within the Dreamweaver application folder, and then call the functions from JavaScript using the JavaScript interpreter that is built into Dreamweaver.

For example, you may want to define a Dreamweaver object that inserts the contents of a user-specified file into the current document. Because client-side JavaScript does not provide any support for file I/O, you must write a function in C to provide this functionality.

You could use the following HTML and JavaScript to create a simple Insert Text from File object. Notice that the `objectTag()` function calls a C function named `readContentsOfFile()`, which is stored in a library named `myLibrary`.

```
<HTML>
<HEAD>
<SCRIPT>
function objectTag() {
    fileName = document.forms[0].myFile.value;
    return myLibrary.readContentsOfFile(fileName);
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
Enter the name of the file to be inserted:
<INPUT TYPE="file" NAME="myFile">
</FORM>
</BODY>
</HTML>
```

The `readContentsOfFile()` function accepts a list of arguments from the user, unpacks the argument containing the file name, reads the contents of the file, and packages the contents of the file as the return value. For more information about the JavaScript data structures and functions that appear in `readContentsOfFile()`, see “The C-level extensibility API” on page 211.

```
JSBool
readContentsOfFile(JSContext *cx, JSObject *obj, unsigned int argc, jsval *argv, jsval *rval)
{
    char *fileName, *fileContents;
    JSBool success;
    unsigned int length;

    /* Make sure caller passed in exactly one argument. If not,
     * then tell the interpreter to abort script execution. */
    if (argc != 1){
        JS_ReportError(cx, "Wrong number of arguments", 0);
        return JS_FALSE;
    }

    /* Convert the argument to a string */
    fileName = JS_ValueToString(cx, argv[0], &length);
    if (fileName == NULL){
        JS_ReportError(cx, "The argument must be a string", 0);
        return JS_FALSE;
    }

    /* Use the string (the file name) to open and read a file */
    fileContents = exerciseLeftToTheReader(fileName);

    /* Store file contents in rval, which is the return value passed
     * back to the caller */
    success = JS_StringToValue(cx, fileContents, 0, *rval);
    free(fileContents);

    /* Return true to continue or false to abort the script */
    return success;
}
```

To ensure that the `readContentsOfFile()` function executes as designed rather than causing a JavaScript error, you must register the function with the JavaScript interpreter by including a function called `MM_Init()` in your library. When Dreamweaver loads the library at startup, it calls the `MM_Init()` function to get three pieces of information:

- The JavaScript name of the function
- A pointer to the function
- The number of arguments that the function expects

The `MM_Init()` function for `myLibrary` might look like this:

```
void
MM_Init()
{
    JS_DefineFunction("readContentsOfFile", readContentsOfFile, 1);
}
```

Your library must include exactly one instance of the following macro:

```
/* MM_STATE is a macro that expands to some definitions that are
 * needed to interact with Dreamweaver. This macro must
 * be defined exactly once in your library. */
MM_STATE
```

**Note:** The library can be implemented in either C or C++, but the file containing `MM_Init()` and `MM_STATE` must be implemented in C. The C++ compiler garbles function names, making it impossible for Dreamweaver to find the `MM_Init()` function.

## The C-level extensibility API

The C code in your library must interact with Dreamweaver's JavaScript interpreter at three different times:

- At startup, to register the library's functions.
- When the function is called, to unpack the arguments that are being passed from JavaScript to C.
- Before the function returns, to package the return value.

To accomplish these tasks, the interpreter defines several data types and exposes an API. Definitions for the following data types and functions appear in the file `mm_jsapi.h`. For your library to work properly, you must include `mm_jsapi.h` at the top of each file in your library with the following line:

```
#include "mm_jsapi.h"
```

### **typedef struct JSContext JSContext**

**Description** A pointer to this opaque data type is passed to the C-level function. Some of the functions in the API accept this pointer as one of their arguments.

### **typedef struct JSObject JSObject**

**Description** A pointer to this opaque data type is passed to the C-level function. This data type represents an object, which may be an array object or some other object type.

## **typedef struct jsval jsval**

**Description** An opaque data structure that can contain an integer, or a pointer to a float, string, or object. Some functions in the API can be used to read the values of function arguments by reading the contents of a `jsval`, and some can be used to write the function's return value by writing a `jsval`.

## **typedef enum { JS\_FALSE = 0, JS\_TRUE = 1 } JSBool**

**Description** A simple data type used to store a Boolean value.

## **typedef JSBool (\*JSNative)(JSContext \*cx, JSObject \*obj, unsigned int argc, jsval \*argv, jsval \*rval)**

**Description** The function signature for C-level implementations of JavaScript functions, where:

- `cx` is a pointer to an opaque `JSContext` structure, which must be passed to some of the functions in the JavaScript API. This variable holds the interpreter's execution context.
- `obj` is a pointer to the object in whose context the script executes. While the script is running, the this keyword is equal to this object.
- `argc` is the number of arguments being passed to the function.
- `argv` is a pointer to an array of `jsvals`. The array is `argc` elements in length.
- `rval` is a pointer to a single `jsval`. The function's return value should be written to `*rval`.

The function returns `JS_TRUE` upon success or `JS_FALSE` upon failure. If `JS_FALSE` is returned, the current script stops executing.

## **JSBool JS\_DefineFunction()**

**Description** Registers a C-level function with the JavaScript interpreter in Dreamweaver. After this function returns, JavaScript scripts that call the function specified in `name` will execute the code pointed to by `call`.

Typically, this function is called from `MM_Init()`, which Dreamweaver calls during startup.

**Arguments** `char *name`, *JSNative call*, unsigned int *nargs*

- `name` is the name of the function as it is exposed to JavaScript.
- `call` is a pointer to a C-level function. The function must accept the same arguments as `readContentsOfFile`, and it must return a `JSBool` indicating success or failure.
- `nargs` is the number of arguments that the function expects to receive.

**Returns** A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

### **char \*JS\_ValueToString()**

<b>Description</b>	Extracts a function argument from a jsval, converts it to a string (if possible), and passes the converted value back to the caller.
<b>Arguments</b>	<p>JSContext *cx, jsval v, unsigned int *pLength</p> <ul style="list-style-type: none"><li>• cx is the opaque JSContext pointer that was passed to the JavaScript function.</li><li>• v is the jsval from which the string is to be extracted.</li><li>• pLength is a pointer to an unsigned integer. This function sets *pLength equal to the length of the string in bytes.</li></ul>
<b>Returns</b>	A pointer to a string on success or NULL on failure.

### **JSBool JS\_ValueToInteger()**

<b>Description</b>	Extracts a function argument from a jsval, converts it to an integer (if possible), and passes the converted value back to the caller.
<b>Arguments</b>	<p>JSContext *cx, jsval v, long *lp</p> <ul style="list-style-type: none"><li>• cx is the opaque JSContext pointer that was passed to the JavaScript function.</li><li>• v is the jsval from which the string is to be extracted.</li><li>• lp is a pointer to a 4-byte long integer. This function stores the converted value in *lp.</li></ul>
<b>Returns</b>	A Boolean value indicating success (JS_TRUE) or failure (JS_FALSE).

### **JSBool JS\_ValueToDouble()**

<b>Description</b>	Extracts a function argument from a jsval, converts it to a double (if possible), and passes the converted value back to the caller.
<b>Arguments</b>	<p>JSContext *cx, jsval v, double *dp</p> <ul style="list-style-type: none"><li>• cx is the opaque JSContext pointer that was passed to the JavaScript function.</li><li>• v is the jsval from which the string is to be extracted.</li><li>• dp is a pointer to an 8-byte double. This function stores the converted value in *dp.</li></ul>
<b>Returns</b>	A Boolean value indicating success (JS_TRUE) or failure (JS_FALSE).

### JSBool JS\_ValueToBoolean()

- Description** Extracts a function argument from a `jsval`, converts it to a Boolean value (if possible), and passes the converted value back to the caller.
- Arguments** `JSContext *cx`, `jsval v`, `JSBool *bp`
- `cx` is the opaque `JSContext` pointer that was passed to the JavaScript function.
  - `v` is the `jsval` from which the string is to be extracted.
  - `bp` is a pointer to a `JSBool`. This function stores the converted value in `*bp`.
- Returns** A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

### JSBool JS\_ValueToObject()

- Description** Extracts a function argument from a `jsval`, converts it to an object (if possible), and passes the converted value back to the caller. If the object is an array, use `JS_GetArrayLength()` and `JS_GetElement()` to read its contents.
- Arguments** `JSContext *cx`, `jsval v`, `JSObject **op`
- `cx` is the opaque `JSContext` pointer that was passed to the JavaScript function.
  - `v` is the `jsval` from which the string is to be extracted.
  - `op` is a pointer to a (`JSObject *`). This function stores the converted value in `*op`.
- Returns** A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

### JSBool JS\_StringToValue()

- Description** Stores a string return value in a `jsval`.
- Arguments** `JSContext *cx`, `char *bytes`, `size_t sz`, `jsval *vp`
- `cx` is the opaque `JSContext` pointer that was passed to the JavaScript function.
  - `bytes` is the string to be stored in the `jsval`. The string data is copied, so the caller should free the string when it is no longer needed. If the string size is not specified (see the `sz` argument), then the string must be null terminated.
  - `sz` is the size of the string, in bytes. If `sz` is 0, then the length of the null-terminated string is computed automatically.
  - `vp` is a pointer to the `jsval` into which the contents of the string should be copied.
- Returns** A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

### JSBool JS\_DoubleToValue()

<b>Description</b>	Stores a floating-point number return value in a jsval.
<b>Arguments</b>	<code>JSText *cx</code> , <code>double dv</code> , <code>jsval *vp</code> <ul style="list-style-type: none"><li>• <code>cx</code> is the opaque <code>JSText</code> pointer that was passed to the JavaScript function.</li><li>• <code>dv</code> is an 8-byte floating-point number.</li><li>• <code>vp</code> is a pointer to the <code>jsval</code> into which the contents of the double should be copied.</li></ul>
<b>Returns</b>	A Boolean value indicating success ( <code>JS_TRUE</code> ) or failure ( <code>JS_FALSE</code> ).

### JSBool JS\_IntegerToValue()

<b>Description</b>	Stores an integer return value in a jsval.
<b>Arguments</b>	<code>long lv</code>
<b>Returns</b>	A Boolean value indicating success ( <code>JS_TRUE</code> ) or failure ( <code>JS_FALSE</code> ).

### JSBool JS\_BooleanToValue()

<b>Description</b>	Stores a Boolean return value in a jsval.
<b>Arguments</b>	<code>JSBool bv</code>
<b>Returns</b>	A Boolean value indicating success ( <code>JS_TRUE</code> ) or failure ( <code>JS_FALSE</code> ).

### JSBool JS\_ObjectToValue()

<b>Description</b>	Stores an object return value in a jsval. Use <code>JS_NewArrayObject()</code> to create an array object, and use <code>JS_SetElement()</code> to define its contents.
<b>Arguments</b>	<code>JSText *obj</code>
<b>Returns</b>	A Boolean value indicating success ( <code>JS_TRUE</code> ) or failure ( <code>JS_FALSE</code> ).

### char \*JS\_ObjectType()

<b>Description</b>	Given an object reference, returns a string describing the type of the object. For array objects, the return value is <code>Array</code> .
<b>Arguments</b>	<code>JSText *obj</code> <p>Typically, this argument is passed in and converted using <code>JS_ValueToObject()</code>.</p>
<b>Returns</b>	A pointer to a null-terminated string. The caller should <i>not</i> free this string when it has finished.

## **JSObject \*JS\_NewArrayObject()**

<b>Description</b>	Creates a new object that contains an array of jsvals.
<b>Arguments</b>	<i>JSContext *cx</i> , unsigned int <i>length</i> , jsval * <i>v</i> <ul style="list-style-type: none"><li>• <i>cx</i> is the opaque JSContext pointer that was passed to the JavaScript function.</li><li>• <i>length</i> is the number of elements that the array will hold.</li><li>• <i>v</i> is an optional pointer to the jsvals to be stored in the array. If the return value is not NULL, <i>v</i> is an array containing <i>length</i> elements. If the return value is NULL, then the initial content of the array object is undefined (and may be set using <i>JS_SetElement()</i>).</li></ul>
<b>Returns</b>	A pointer to a new array object, or NULL upon failure.

## **long JS\_GetArrayLength()**

<b>Description</b>	Given a pointer to an array object, gets the number of elements in the array.
<b>Arguments</b>	<i>JSContext *cx</i> , JSObject * <i>obj</i> <ul style="list-style-type: none"><li>• <i>cx</i> is the opaque JSContext pointer that was passed to the JavaScript function.</li><li>• <i>obj</i> is a reference to an array object.</li></ul>
<b>Returns</b>	The number of elements in the array, or -1 upon failure.

## **JSBool JS\_GetElement()**

<b>Description</b>	Reads a single element of an array object.
<b>Arguments</b>	<i>JSContext *cx</i> , JSObject * <i>obj</i> , unsigned int <i>index</i> , jsval * <i>v</i> <ul style="list-style-type: none"><li>• <i>cx</i> is the opaque JSContext pointer that was passed to the JavaScript function.</li><li>• <i>obj</i> is a pointer to an array object.</li><li>• <i>index</i> is an integer index into the array. The first element is index 0, and the last element is index (<i>length</i> - 1).</li><li>• <i>v</i> is a pointer to a jsval where the contents of the jsval in the array should be copied.</li></ul>
<b>Returns</b>	A Boolean value indicating success ( <i>JS_TRUE</i> ) or failure ( <i>JS_FALSE</i> ).



## JSBool JS\_SetElement()

<b>Description</b>	Writes a single element of an array object.
<b>Arguments</b>	<i>JSContext *cx</i> , <i>JSObject *obj</i> , unsigned int <i>index</i> , jsval * <i>v</i> <ul style="list-style-type: none"><li>• <i>cx</i> is the opaque <i>JSContext</i> pointer that was passed to the JavaScript function.</li><li>• <i>obj</i> is a pointer to an array object.</li><li>• <i>index</i> is an integer index into the array. The first element is index 0, and the last element is index (length - 1).</li><li>• <i>v</i> is a pointer to a <i>jsval</i> whose contents should be copied to the <i>jsval</i> in the array.</li></ul>
<b>Returns</b>	A Boolean value indicating success ( <i>JS_TRUE</i> ) or failure ( <i>JS_FALSE</i> ).

## JSBool JS\_ExecuteScript()

<b>Description</b>	Compiles and executes a string of Javascript. If the script generates a return value, it is returned in * <i>rval</i> .
<b>Arguments</b>	<i>JSContext *cx</i> , <i>JSObject *obj</i> , char * <i>script</i> , unsigned int <i>sz</i> , jsval * <i>rval</i> <ul style="list-style-type: none"><li>• <i>cx</i> is the opaque <i>JSContext</i> pointer that was passed to the JavaScript function.</li><li>• <i>obj</i> is a pointer to the object in whose context the script executes. While the script is running, the <i>this</i> keyword is equal to this object. Usually this is the <i>JSObject</i> pointer that was passed to the JavaScript function.</li><li>• <i>script</i> is a string containing JavaScript code. If the string size is not specified (see the <i>sz</i> argument), then the string must be null-terminated.</li><li>• <i>sz</i> is the size of the string, in bytes. If <i>sz</i> is 0, then the length of the null-terminated string is computed automatically.</li><li>• <i>rval</i> is a pointer to a single <i>jsval</i>. The function's return value is stored in *<i>rval</i>.</li></ul>
<b>Returns</b>	A Boolean value indicating success ( <i>JS_TRUE</i> ) or failure ( <i>JS_FALSE</i> ).

## JSBool JS\_ReportError()

<b>Description</b>	Describes the reason for a script error. Call this function before returning <code>JS_FALSE</code> to give the user information about why the script failed (for example, “wrong number of arguments”).
<b>Arguments</b>	<code>JSContext *cx</code> , <code>char *error</code> , <code>size_t sz</code> <ul style="list-style-type: none"><li>• <code>cx</code> is the opaque <code>JSContext</code> pointer that was passed to the JavaScript function.</li><li>• <code>error</code> is a string containing the error message. The string is copied, so the caller should free the string when it is no longer needed. If the string size is not specified (see <code>sz</code>, below), then the string must be null-terminated.</li><li>• <code>sz</code> is the size of the string, in bytes. If <code>sz</code> is 0, then the length of the null-terminated string is computed automatically.</li></ul>
<b>Returns</b>	A Boolean value indicating success ( <code>JS_TRUE</code> ) or failure ( <code>JS_FALSE</code> ).

## Calling a C function from JavaScript

Once you know how C-level extensibility works in Dreamweaver and the data types and functions it relies on, it's useful to walk through an example of how to build a library and call a function.

This exercise requires three files, which are included in the `Extending/c_files` folder inside the Dreamweaver application folder:

- `mm_jsapi.h` is a header file that includes definitions for all the data types and functions described in “The C-level extensibility API” on page 211
- `Sample.c` is an example file that defines the `computeSum()` function.
- `Sample.mak` is a makefile that you can use to build `Sample.c` into a DLL with Microsoft Visual C++; `Sample.proj` is the equivalent file for building a CFM Library with Metrowerks CodeWarrior. If you are using another tool, you can create the makefile yourself.

### To build the DLL in Windows:

- 1 In Microsoft Visual C++, choose `File > Open Workspace` and select `Sample.mak`.
- 2 Choose `Build > Rebuild All`.

When the build operation is complete, a file called `Sample.dll` appears in the folder containing `Sample.mak` (or one of its subfolders).

**To build the shared library on the Macintosh:**

- 1 Open Sample.proj in Metrowerks CodeWarrior.
- 2 Build the project to generate a CFM Library.

When the build operation has finished, a file called Sample appears in the folder containing Sample.proj (or in one of its subfolders).

**To call the computeSum() function from the Insert Horizontal Rule object:**

- 1 In the Configuration folder within the Dreamweaver application folder, create a folder called JSExtensions.
- 2 Copy Sample.dll (Windows) or Sample (Macintosh) to the JSExtensions folder.
- 3 In a text editor, open the file called horizontal\_rule.htm that resides in the Configuration/Objects/Common folder.
- 4 Add the line `alert(Sample.computeSum(2,2));` to the `objectTag()` function so that it appears as follows:

```
function objectTag() {  
    // Return the html tag that should be inserted  
    alert(Sample.computeSum(2,2));  
    return "<HR>";  
}
```

- 5 Save the file and restart Dreamweaver.

**To execute the computeSum() function:**

Choose Insert > Horizontal Rule. A dialog box containing the number 4—the result of computing the sum of 2 plus 2—appears.



## CHAPTER 9

### Objects

---

Objects are designed to insert a specific string of code into the user's document. An object appears in a panel on the Object palette and in the Insert menu once its object file is stored in a subfolder within the Configuration/Objects folder inside the Dreamweaver application folder.

Objects have two components: the object file that defines what is inserted in your document; and the 18-pixel by 18-pixel GIF image that appears in the Object palette.

Objects are HTML files. The BODY of an object file can contain an HTML form that accepts parameters for the object (for example, the number of rows and columns in the table to be inserted). The HEAD of an object file contains JavaScript functions that process form input from the BODY and control what is added to the user's document.

**Note:** The simplest objects contain only the HTML to be inserted, with no BODY and HEAD tag. See "Customizing Dreamweaver" in *Using Dreamweaver* for more information.

## How object files work

When a user selects an object by clicking an icon in the Object palette or by choosing an item in the Insert menu, the following chain of events occurs:

- 1 The object file is scanned for a `FORM` tag. If a form exists—and if the Show Dialog When Inserting Objects option is selected in the General preferences—Dreamweaver calls the `windowDimensions()` function, if defined, to determine the size of the dialog box in which to display the form. If no form exists in the object file, Dreamweaver does not display a dialog box, and step 2 is skipped.
- 2 If Dreamweaver displayed a dialog box in step 1, the user enters parameters for the object (such as the number of rows and columns in a table) in the dialog box and clicks OK.
- 3 The `objectTag()` function is called, and its return value is inserted into the document after the current selection (it does not replace the current selection).

## The object API

There are only three custom functions in the object API, and none is required. The functions in the object API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of object files. That is, Dreamweaver automatically calls the `objectTag()` function if it is defined in an object file, whereas in any other extension file a function named `objectTag()` acts like a user-defined function—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions in the main API work: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

## displayHelp()

**Description** If this function is defined, displays a Help button below the OK and Cancel buttons in the parameters dialog box. This function is called when the user clicks the Help button.

**Arguments** None.

**Returns** Nothing.

**Example** The following instance of displayHelp() opens in a browser window a file with instructions for using the object:

```
function displayHelp(){
    dreamweaver.browseDocument('http://people.netscape.com/andrew/dreamweaver/
objects.html');
}
```

## objectTag()

**Description** Inserts a string of code into the user's document.

**Arguments** None.

**Returns** The string to be inserted.

**Example** The following instance of objectTag() inserts an OBJECT/EMBED combination for a specific ActiveX control and plugin:

```
function objectTag() {
    return '\n' +
    '<OBJECT CLASSID="clsid:166F100B-3A9R-11FB-8075444553540000" \n' +
    'CODEBASE="http://www.mysite.com/product/cabs/myproduct.cab#version=1,0,0,0" \n' +
    'NAME="MyProductName"> \n' +
    '<PARAM NAME="SRC" VALUE=""> \n' +
    '<EMBED SRC="" HEIGHT="" WIDTH="" NAME="MyProductName"> \n' +
    '</OBJECT>'
```

## windowDimensions()

<b>Description</b>	<p>Sets specific dimensions for the parameters dialog box. If this function is not defined, the window dimensions are computed automatically.</p> <p><b>Note:</b> Do not define this function unless you want an options dialog box larger than 640 pixels by 480 pixels.</p>
<b>Arguments</b>	<p><i>platform</i></p> <p>The value of the argument is either "macintosh" or "windows", depending on the user's platform.</p>
<b>Returns</b>	<p>A string of the form "widthInPixels,heightInPixels".</p> <p>The returned dimensions are smaller than the size of the entire dialog window because they do not include the area for the OK and Cancel buttons. If the returned dimensions do not accommodate all options, scroll bars appear.</p>
<b>Example</b>	<p>The following instance of windowDimensions() sets the dimensions of the parameters dialog box to 648 pixels by 520 pixels:</p> <pre>function windowDimensions(platform){     return "648,520"; }</pre>

## Adding objects to the Object palette

Dreamweaver automatically adds any files that are inside one of the subfolders in the Configuration/Objects folder to the panel associated with the subfolder. For example, a file inside the Configuration/Objects/MyObjects folder would appear on the MyObjects panel of the Object palette.

**Note:** Although object files can be stored in separate folders, it's important that their file names be unique. The dom.insertObject() function, for example, looks for a specified file anywhere within the Objects folder without regard to subfolders. If a file called Button.htm exists in the Forms folder and also in the MyObjects folder, Dreamweaver cannot distinguish between them.

Each object file has an associated 18-pixel by 18-pixel GIF image that appears in the Object palette. The image file must have the same base name as the object file (for example, object.gif is associated with object.htm) to ensure that the files remain connected.

If you create a larger object image, Dreamweaver will scale it to 18 pixels by 18 pixels. If you do not create an image for your object, a broken image icon appears in the Object palette.



## Adding objects to the Insert menu

Dreamweaver automatically adds to the bottom of the Insert menu any files that are inside one of the subfolders in the Configuration/Objects folder.

To control the position of an object in the Insert menu or any other menu, or to add an object to multiple menus, you can modify the menus.xml file. In Dreamweaver 3, this file controls the entire menu structure for Dreamweaver. For more information about modifying the menus.xml file, see Chapter 16, “Customizing Dreamweaver” in *Using Dreamweaver*.

**Note:** In previous versions of Dreamweaver, the items in the Insert menu were controlled by a file called InsertMenu.htm. This file has been superseded by menus.xml.



## CHAPTER 10

### Commands

---

Commands can be used to perform almost any kind of edit to the user's current document, other open documents, or to any HTML document on a local drive. Commands can insert, remove, or rearrange HTML tags and attributes, comments, and text.

Commands are HTML files. The `BODY` of a command file can contain an HTML form that accepts options for the command (for example, how a table should be sorted and by which column). The `HEAD` of a command file contains JavaScript functions that process form input from the `BODY` and control what edits are made to the user's document.

## How commands work

When a user clicks a menu that contains a command, the following chain of events occurs:

- 1 Dreamweaver calls the `canAcceptCommand()` function, if defined, in each command file referenced in the menu to see whether this command is appropriate for the selection. If `canAcceptCommand()` returns `FALSE`, the command is dimmed in the menu.
- 2 The user selects a command from the menu.
- 3 Dreamweaver calls the `receiveArguments()` function, if defined, in the selected command file to give the command an opportunity to process any arguments passed from the `dreamweaver.runCommand()` function.
- 4 Dreamweaver calls the `commandButtons()` function, if defined, to determine which buttons appear along the right side of the options dialog box and what code should be executed when the user clicks the buttons.
- 5 Dreamweaver scans the command file for a `FORM` tag. If a form exists, Dreamweaver calls the `windowDimensions()` function to determine the size of the options dialog box containing the `BODY` elements of the file. If `windowDimensions()` is not defined, the size of the dialog box is determined automatically.
- 6 If the command file's `BODY` tag contains an `onLoad` handler, Dreamweaver executes it (regardless of whether a dialog box is displayed). If no dialog box appears, the remaining steps do not occur.
- 7 The user selects options for the command. Dreamweaver executes event handlers associated with the fields as the user encounters them.
- 8 The user clicks one of the buttons defined by `commandButtons()`.
- 9 Dreamweaver executes the code associated with the button that was clicked.
- 10 The dialog box remains visible until one of the scripts in the command calls `window.close()`.

## The command API

There are four custom functions in the command API, and none is required. The functions in the command API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of command files. That is, Dreamweaver automatically calls the `commandButtons()` function if it is defined in a command or menu command file, whereas in any other extension file a function named `commandButtons()` acts like a user-defined function—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions in the main API work: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

### `canAcceptCommand()`

<b>Description</b>	Determines whether the command is appropriate for the current selection.  <b>Note:</b> Do not define <code>canAcceptCommand()</code> unless it returns <code>FALSE</code> in at least one case. If the function is not defined, the command is assumed to be appropriate; making this assumption saves time and improves performance.
<b>Arguments</b>	None.
<b>Returns</b>	<code>TRUE</code> if the command is allowed; <code>FALSE</code> if it is not, dimming the command in the menu.
<b>Example</b>	The following instance of <code>canAcceptCommand()</code> makes the command available only when the selection is a table:  <pre>function canAcceptCommand(){     var selArr=dreamweaver.getSelection();     var selObj=dreamweaver.offsetsToNode(selArr[0],selArr[1]);     return (selObj.nodeType == Node.ELEMENT_NODE &amp;&amp; selObj.tagName=="TABLE"); }</pre>

## commandButtons()

**Description** Defines the buttons that should appear on the right side of the options dialog box and their behavior when clicked. If this function is not defined, no buttons appear, and the BODY of the command file expands to fill the entire dialog box.

**Arguments** None.

**Returns** An array containing an even number of elements. The first element is a string containing the label for the topmost button. The second element is a string of JavaScript code that defines the behavior of the topmost button when clicked. Remaining elements define additional buttons in the same manner.

**Example** The following instance of commandButtons() defines three buttons: OK, Cancel, and Help.

```
function commandButtons(){  
    return new Array("OK","doCommand()","Cancel","window.close()","Help","showHelp()");  
}
```

## receiveArguments()

**Description** Processes any arguments passed from the menu item.

**Arguments** {arg1}, {arg2},...{argN}

If the arguments attribute is defined for a menuitem tag, the value of that attribute is passed to the receiveArguments() function as one or more arguments. The arguments attribute is useful for distinguishing between two menu items that call the same menu command.

**Returns** Nothing.

## windowDimensions()

**Description** Sets specific dimensions for the parameters dialog box to speed up display. If this function is not defined, the window dimensions are computed automatically.

**Note:** Do not define this function unless you want an options dialog box larger than 640 pixels by 480 pixels.

**Arguments** *platform*

The value of the argument is either "macintosh" or "windows", depending on the user's platform.

**Returns** A string of the form "widthInPixels,heightInPixels".

The returned dimensions are smaller than the size of the entire dialog window because they do not include the area for the OK and Cancel buttons. If the returned dimensions do not accommodate all options, scroll bars appear.

**Example** The following instance of windowDimensions() sets the dimensions of the parameters dialog box to 648 pixels by 520 pixels:

```
function windowDimensions(platform){  
  return "648,520";  
}
```

## A simple command example

The following command converts the selected text to all lowercase characters. The command is very simple; it does not display a dialog box, so the `commandButtons()` function is not defined.

```
<HTML>
<HEAD>
<TITLE>Make Lower Case</TITLE>
<SCRIPT LANGUAGE="javascript">

function canAcceptCommand(){
    // Get the DOM of the current document
    var theDOM = dw.getDocumentDOM();
    // Get the offsets of the selection
    var theSel = theDOM.getSelection();
    // Get the selected node
    var theSelNode = theDOM.getSelectedNode();
    // Get the children of the selected node
    var theChildren = theSelNode.childNodes;

    // If the selection is not an insertion point, and
    // either the selection or its first child is a
    // text node, return TRUE.
    return (theSel[0] != theSel[1] && (theSelNode.nodeType == Node.TEXT_NODE ||
theChildren[0].nodeType == Node.TEXT_NODE));
}

function changeToLowerCase() {
    // Get the DOM again
    var theDOM = dw.getDocumentDOM();
    // Get the offsets of the selection
    var theSel = theDOM.getSelection();

    // Get the outerHTML of the HTML tag (the
    // entire contents of the document)
    var theDocEl = theDOM.documentElement;
    var theWholeDoc = theDocEl.outerHTML;

    // Extract the selection
    var selText = theWholeDoc.substring(theSel[0],theSel[1]);

    // Re-insert the modified selection into the document
    theDocEl.outerHTML = theWholeDoc.substring(0,theSel[0]) + selText.toLowerCase() +
theWholeDoc.substring(theSel[1]);

    // Set the selection back to where it was when you
    // started
    theDOM.setSelection(theSel[0],theSel[1]);
}

</SCRIPT>
```



```
</HEAD>
```

```
<BODY onLoad="changeToLowerCase()">
```

<!-- The function that does all the work in this command is called from the onLoad handler on the BODY tag. There is no form in the BODY, so no dialog box appears. -->

```
</BODY>
```

```
</HTML>
```

## Adding commands to the Commands menu

Dreamweaver automatically adds to the bottom of the Commands menu any files that are inside one of the subfolders in the Configuration/Objects folder. To prevent a command from appearing in the Commands menu, put the following comment on the first line of the file:

```
<!-- MENU-LOCATION=NONE -->
```

In Dreamweaver 1 and 2, if you wanted to control the position or text associated with a command, you could add it explicitly to the CommandMenu.htm file. In Dreamweaver 3, this file has been superseded by the menus.xml file, which controls the entire menu structure for Dreamweaver. For more information about modifying the menus.xml file, see Chapter 16, “Customizing Dreamweaver” in *Using Dreamweaver*.



## CHAPTER 11

### Menu Commands

---

Dreamweaver 3 introduces a new type of command that makes menus more flexible and dynamic menus possible. Like regular commands, menu commands can be used to perform almost any kind of edit to the current document, other open documents, or any HTML document on a local drive. The API for menu commands expands the regular command API to accomplish several tasks related to displaying and calling the command from the menu system.

**Note:** Because menu commands are directly related to the menu system in Dreamweaver, you should read Chapter 16, "Customizing Dreamweaver" in *Using Dreamweaver* before continuing.

Menu commands are HTML files. The BODY of a menu command file can contain an HTML form that accepts options for the command (for example, how a table should be sorted and by which column). The HEAD of a menu command file contains JavaScript functions that process form input from the BODY and control what edits are made to the user's document.

Menu commands are stored in the Configuration/Menus folder inside the Dreamweaver application folder.

## How menu commands work

When the user clicks a menu that contains a menu command, the following chain of events occurs:

- 1 If any `menuitem` tag in the menu contains the dynamic attribute, Dreamweaver calls the `getDynamicContent()` function in the associated menu command file to populate the menu.
- 2 Dreamweaver calls the `canAcceptCommand()` function in each menu command file referenced in the menu to check whether the command is appropriate for the selection. If `canAcceptCommand()` returns `FALSE`, the menu item is dimmed. If `canAcceptCommand()` returns `TRUE` or is not defined, Dreamweaver calls the `isCommandChecked()` function to determine whether to display a check mark next to the menu item. If the `isCommandChecked()` function is not defined, no check mark is displayed.
- 3 Dreamweaver calls the `setMenuText()` function to determine the text that should appear in the menu. If `setMenuText()` is not defined, Dreamweaver uses the text that is specified in the `menuitem` tag.
- 4 The user selects an item from the menu.
- 5 Dreamweaver calls the `receiveArguments()` function, if defined, in the selected menu command file to let the command process any arguments passed from the menu item.
- 6 Dreamweaver calls the `commandButtons()` function, if defined, to determine which buttons appear along the right side of the options dialog box and what code should be executed when the user clicks the buttons.
- 7 Dreamweaver scans the menu command file for a `FORM` tag. If a form exists, Dreamweaver calls the `windowDimensions()` function to determine the size of the options dialog box containing the `BODY` elements of the file. If `windowDimensions()` is not defined, the size of the dialog box is determined automatically.
- 8 If the menu command file's `BODY` tag contains an `onLoad` handler, Dreamweaver executes the code associated with the handler (regardless of whether a dialog box is displayed). If no dialog box appears, the remaining steps do not occur.
- 9 The user selects options in the dialog box. Dreamweaver executes event handlers associated with the fields as the user encounters them.
- 10 The user clicks one of the buttons defined by `commandButtons()`.
- 11 Dreamweaver executes the code associated with the clicked button.
- 12 The dialog box remains visible until one of the scripts in the menu command calls `window.close()`.

## The menu command API

There are seven custom functions in the menu command API, and none is required. The functions in the menu command API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of menu command files. That is, Dreamweaver automatically calls the `getDynamicContent()` function if it is defined in a menu command file, whereas in any other extension file a function named `getDynamicContent()` acts like a user-defined function—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions in the main API work: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

### `canAcceptCommand()`

<b>Description</b>	Determines whether the menu item should be active or dimmed.
<b>Arguments</b>	<code>{arg1}, {arg2},...{argN}</code>  If the <code>arguments</code> attribute is defined for a <code>menuitem</code> tag, the value of that attribute is passed to the <code>canAcceptCommand()</code> function (and to the <code>isCommandChecked()</code> , <code>receiveArguments()</code> , and <code>setMenuText()</code> functions) as one or more arguments. The <code>arguments</code> attribute is useful for distinguishing between two menu items that call the same menu command.
<b>Returns</b>	A Boolean value indicating whether the item should be enabled.

### `commandButtons()`

<b>Description</b>	Defines the buttons that should appear on the right side of the options dialog box and their behavior when clicked. If this function is not defined, no buttons appear, and the <code>BODY</code> of the command file expands to fill the entire dialog box.
<b>Arguments</b>	None.
<b>Returns</b>	An array containing an even number of elements. The first element is a string containing the label for the topmost button. The second element is a string of JavaScript code that defines the behavior of the topmost button when clicked. Remaining elements define additional buttons in the same manner.
<b>Example</b>	The following instance of <code>commandButtons()</code> defines three buttons: OK, Cancel, and Help.  <pre>function commandButtons(){     return new Array("OK","doCommand()","Cancel","window.close()","Help","showHelp()"); }</pre>

## getDynamicContent()

<b>Description</b>	Retrieves the content for the dynamic portion of the menu.
<b>Arguments</b>	<i>menuID</i>  The argument is the value of the id attribute in the menuitem tag associated with the item.
<b>Returns</b>	An array of strings. Each string contains the name of a menu item and its unique ID, separated by a semicolon. If the function returns null, then no change is made to the menu.
<b>Example</b>	The following instance of getDynamicContent() returns an array of four menu items (My Menu Item 1, My Menu Item 2, and so on):  <pre>function getDynamicContent(){   var stringArray= new Array();   var i=0;   var numItems = 4;    for (i=0; i&lt;numItems;i++)     stringArray[i] = new String("My Menu Item " + i + ";id=" + i);    return stringArray; }</pre>

## isCommandChecked()

<b>Description</b>	Determines whether to display a check mark next to the menu item.
<b>Arguments</b>	<i>{arg1}, {arg2},...{argN}</i>  If the arguments attribute is defined for a menuitem tag, the value of that attribute is passed to the isCommandChecked() function (and to the canAcceptCommand(), receiveArguments(), and setMenuText() functions) as one or more arguments. The arguments attribute is useful for distinguishing between two menu items that call the same menu command.
<b>Returns</b>	A Boolean value indicating whether a check mark should appear next to the menu item.

## receiveArguments()

<b>Description</b>	Processes any arguments passed from the menu item.
<b>Arguments</b>	<i>{arg1}, {arg2},...{argN}</i>  If the arguments attribute is defined for a menuitem tag, the value of that attribute is passed to the receiveArguments() function (and to the canAcceptCommand(), isCommandChecked(), and setMenuText() functions) as one or more arguments. The arguments attribute is useful for distinguishing between two menu items that call the same menu command.
<b>Returns</b>	Nothing.

## setMenuText()

**Description** Specifies the text that should appear in the menu.

**Note:** Do not use this function if you are using `getDynamicContent()`.

**Arguments** `{arg1}, {arg2},...{argN}`

If the `arguments` attribute is defined for a `menuitem` tag, the value of that attribute is passed to the `setMenuText()` function (and to the `canAcceptCommand()`, `isCommandChecked()`, and `receiveArguments()` functions) as one or more arguments. The `arguments` attribute is useful for distinguishing between two menu items that call the same menu command.

**Returns** The string that should appear in the menu.

## windowDimensions()

**Description** Sets specific dimensions for the parameters dialog box to speed up display. If this function is not defined, the window dimensions are computed automatically.

**Note:** Do not define this function unless you want an options dialog box larger than 640 pixels by 480 pixels.

**Arguments** `platform`

The value of the argument is either "macintosh" or "windows", depending on the user's platform.

**Returns** A string of the form "widthInPixels,heightInPixels".

The returned dimensions are smaller than the size of the entire dialog window because they do not include the area for the OK and Cancel buttons. If the returned dimensions do not accommodate all options, scroll bars appear.

**Example** The following instance of `windowDimensions()` sets the dimensions of the parameters dialog box to 648 pixels by 520 pixels:

```
function windowDimensions(platform){  
    return "648,520";  
}
```

## A simple menu command

The following menu command is associated with two menu items: Undo and Redo. It checks the arguments attribute of the menuitem tag and performs a `dw.undo()` or a `dw.redo()` operation depending on the value of the first (and only) argument.

```
<HTML>
<HEAD>
<!-- Copyright 1999 Macromedia, Inc. All rights reserved. -->
<TITLE>Edit Clipboard</TITLE>
<SCRIPT LANGUAGE="javascript">
function receiveArguments()
{
    if (arguments.length != 1) return;

    var whatToDo = arguments[0];
    if (whatToDo == "undo")
    {
        dw.undo();
    }
    else if (whatToDo == "redo")
    {
        dw.redo();
    }
}

function canAcceptCommand()
{
    var selarray;
    if (arguments.length != 1) return false;
    var bResult = false;

    var whatToDo = arguments[0];
    if (whatToDo == "undo")
    {
        bResult = dw.canUndo();
    }
    else if (whatToDo == "redo")
    {
        bResult = dw.canRedo();
    }
    return bResult;
}

function setMenuText()
{
    if (arguments.length != 1) return "";

    var whatToDo = arguments[0];
    if (whatToDo == "undo")
        return dw.getUndoText();
```



```

else if (whatToDo == "redo")
    return dw.getRedoText();
else return "";
}

```

```

</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>

```

In this command, the `receiveArguments()` function both processes the arguments and executes the command, but this need not be the case. More complex menu commands might call different functions to execute the command. For example, the following code checks whether the first argument is "foo"; if it is, it calls the `doOperationX()` function and passes it the second argument. If the first argument is "bar", it calls the `doOperationY()` function and passes it the second argument. `doOperationX()` or `doOperationY()` is responsible for executing the command.

```

function receiveArguments(){
    if (arguments.length != 2) return;

    var whatToDo = arguments[0];

    if (whatToDo == "foo"){
        doOperationX(arguments[1]);
    }else if (whatToDo == "bar"){
        doOperationY(arguments[1]);
    }
}

```

## A simple dynamic menu

The following menu command does two things: it generates the Preview in Browser submenu, and it launches the current file (or the selected files in the Site window) in the browser that the user selects from the submenu.

```
<HTML>
<HEAD>
<!-- Copyright 1999 Macromedia, Inc. All rights reserved. -->
<TITLE>Preview Browsers</TITLE>
<SCRIPT LANGUAGE="javascript">
<!--
// getDynamicContent returns the contents of a dynamically generated menu.
// returns an array of strings to be placed in the menu, with a unique
// identifier for each item separated from the menu string by a semicolon.
//
// return null from this routine to indicate that you are not adding any
// items to the menu
function getDynamicContent(itemID)
{
    var browsers = null;
    var PIB = null;
    var i;
    var j=0;
    var bUpdate = dw.getMenuNeedsUpdating(itemID);

    if (bUpdate)
    {
        browsers = new Array();
        PIB = dw.getBrowserList();
        // each browser pair has the name of the browser and the path that leads
        // to the application on disk. We only put the names in the menus.
        for (i=0; i<PIB.length; i=i+2)
        {
            browsers[j] = new String(PIB[i]);

            if (dw.getPrimaryBrowser() == PIB[i+1])
                browsers[j] += "\tF12";
            if (navigator.platform == "MacPPC")
            {
                if (dw.getSecondaryBrowser() == PIB[i+1])
                    browsers[j] += "\t ?F12";
            }
            else
            {
                if (dw.getSecondaryBrowser() == PIB[i+1])
                    browsers[j] += "\t Ctrl+F12";
            }

            browsers[j] += ";id=""+PIB[i] +""";
            j = j+1;
        }
    }
}
```

```

        dw.notifyMenuUpdated(itemID, "dw.getBrowserList()");
    }
    return browsers;
}

function canAcceptCommand()
{
    var bHaveDocument;

    if (dw.getFocus() == 'site')
        bHaveDocument = site.getSelection().length > 0;
    else
        bHaveDocument = dw.getDocumentDOM('document') != null;

    return bHaveDocument;
}

function receiveArguments()
{
    var theBrowser = arguments[0];
    if (dw.getFocus() == 'site')
        dw.browseDocument(site.getSelection(),theBrowser);
    else
        dw.browseDocument(dw.getDocumentPath('document'),theBrowser);
}

// -->
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>

```



## CHAPTER 12

### Property Inspectors

.....

The Property inspector is perhaps the most familiar floating palette in the Dreamweaver interface. It is indispensable for defining, reviewing, and changing the name, size, appearance, and other attributes of the selection, as well as for launching internal and external editors for the selected element.

Dreamweaver has several built-in interfaces for the Property inspector that let you set properties for many standard HTML tags. With custom property inspector files, you can override these built-in interfaces or create new ones to inspect custom tags.

Property inspector files are HTML files that reside in the Configuration/Inspectors folder inside the Dreamweaver application folder. The first line of a property inspector file (the line above the opening HTML tag) must be a comment in the following format:

```
<!-- tag:tagNameOrKeyword,priority:1to10,selection:exactOrWithin,hline,vline -->
```

where:

- *tagNameOrKeyword* is the tag to be inspected or one of the following keywords: \*COMMENT\* (for comments), \*LOCKED\* (for locked regions), or \*ASP\* (for ASP tags).
- *1to10* is the priority of the inspector file: 1 indicates that this inspector should be used only when no others can inspect the selection; 10 indicates that this inspector takes precedence over all others that can inspect the selection.
- *exactOrWithin* indicates whether the selection can be within the tag (within) or must exactly contain the tag (exact).
- *hline* (optional) indicates that a horizontal gray line should appear between the upper and lower halves of the inspector in expanded mode.
- *vline* (optional) indicates that a vertical gray line should appear between the tag name field and the rest of the properties in the inspector (see the image property inspector for an example).

The following comment would be appropriate for an inspector that is designed to inspect the HAPPY tag:

```
<!-- tag:HAPPY,priority:8,selection:exact,hline,vline -->
```

The BODY of a property inspector file contains an HTML form. Instead of displaying the form contents in a dialog box, however, Dreamweaver uses the form to define the input areas and layout of the inspector.

## How property inspector files work

At startup, Dreamweaver reads the first line of each .htm and .html file in the Configuration/Inspectors folder, looking for the comment string that defines the type, priority, and selection type of a property inspector. Files that do not have this comment as their first line are ignored.

When the user makes a selection in Dreamweaver or moves the insertion point to a different location, the following chain of events occurs:

- 1 Dreamweaver looks for any inspectors that have a selection type of within.
- 2 If there are any within inspectors, Dreamweaver searches up the document tree from the currently selected tag to check whether there are inspectors for any of the tags that surround the selection. If—and only if—there are no within inspectors, Dreamweaver looks for any inspectors that have a selection type of exact.
- 3 For the first tag found that has one or more inspectors, Dreamweaver calls each inspector's `canInspectSelection()` function. If this function returns `FALSE`, Dreamweaver no longer considers the inspector a candidate for inspecting the selection.
- 4 If more than one potential inspector remains after calling `canInspectSelection()`, Dreamweaver sorts the remaining inspectors by priority.
- 5 If more than one potential inspector shares the same priority, Dreamweaver chooses an inspector alphabetically by name.
- 6 The chosen inspector appears in the Property inspector floating palette. If the property inspector file defines the `displayHelp()` function, a small ? icon is displayed in the upper right corner of the inspector.
- 7 Dreamweaver calls the `inspectSelection()` function to gather information about the current selection and populate the inspector's fields.
- 8 Event handlers attached to the fields in the property inspector interface execute as the user encounters them. (For example, you may have an `onBlur` event that calls `setAttribute()` to set an attribute to the value just entered by the user.)

## The property inspector API

There are three custom functions in the property inspector API, two of which (`canInspectSelection()` and `inspectSelection()`) are required. The functions in the property inspector API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of property inspector files. That is, Dreamweaver automatically calls the `canInspectSelection()` function in a property inspector file, whereas in any other extension file a function named `canInspectSelection()` acts like a user-defined function—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions in the main API work: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

### `canInspectSelection()`

<b>Description</b>	Determines whether the property inspector is appropriate for the current selection.
<b>Arguments</b>	None.  Use <code>dom.getSelectedNode()</code> to get the current selection as a JavaScript object.
<b>Returns</b>	TRUE if the inspector can inspect the current selection; FALSE if it cannot.
<b>Example</b>	<p>The following instance of <code>canInspectSelection()</code> returns TRUE if the selection contains the <code>CLASSID</code> attribute and the value of that attribute is <code>clsid:D27CDB6E-AE6D-11cf-96B8-444553540000</code> (the class ID for Flash Player):</p> <pre>function canInspectSelection(){     var theDOM = dw.getDocumentDOM();     var currSel = theDOM.getSelection();     var theObj = theDOM.offsetsToNode(currSel[0],currSel[1]);     return (theObj.nodeType == Node.ELEMENT_NODE &amp;&amp; theObj.hasAttribute("classid") &amp;&amp;     theObj.getAttribute("classid").toLowerCase()=="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"); }</pre>

## displayHelp()

<b>Description</b>	If this function is defined, a ? icon appears in the upper right corner of the property inspector. This function is called when the user clicks the icon.
<b>Arguments</b>	None.
<b>Returns</b>	Nothing.
<b>Example</b>	<p>The following instance of displayHelp() opens in a browser window a file that explains the fields in the property inspector:</p> <pre>function displayHelp(){     dreamweaver.browseDocument('http://www.hooha.com/dw/inspectors/inspHelp.html'); }</pre>

## inspectSelection()

<b>Description</b>	Refreshes the contents of the user-input fields based on the attributes of the current selection.
<b>Arguments</b>	<p><i>maxOrMin</i></p> <p>The argument is either max or min, depending on whether the inspector is in its expanded or contracted state.</p>
<b>Returns</b>	Nothing.
<b>Example</b>	<p>The following instance of inspectSelection() gets the value of the CONTENT attribute and uses it to populate a form field called keywords:</p> <pre>function inspectSelection(){     var currSel = dreamweaver.getSelection();     var theObj = dreamweaver.offsetsToNode(currSel[0],currSel[1]);     document.forms[0].keywords.value = theObj.getAttribute("content"); }</pre>



## A simple property inspector example

The following property inspector inspects a fictional tag called INTJ. The INTJ tag is empty (that is, it has no closing tag), so its selection type is exact. As long as the selection is exactly an INTJ tag, the inspector should show up—so the `canInspectSelection()` function returns `TRUE` every time. To have a different inspector appear depending on the value of the INTJ tag's `TYPE` attribute, for example, the `canInspectSelection()` function must check the value of the `TYPE` attribute to determine which inspector is the right one. (This is how the Keywords and Description inspectors work, because “keywords” and “description” are not tags but values of the META tag's `NAME` attribute.)

```
<!-- tag:INTJ,priority:5,selection:exact,vline,hline -->
<HTML>
<HEAD>
<TITLE>Interjection Inspector</TITLE>
<SCRIPT LANGUAGE="JavaScript">

function canInspectSelection(){
    return true;
}

function inspectSelection(){
    // Get the DOM of the current document
    var theDOM = dw.getDocumentDOM();
    // Get the selected node
    var theObj = theDOM.getSelectedNode();

    // Get the value of the TYPE attribute on the INTJ tag
    var theType = theObj.getAttribute('type');
    // Initialize a variable called typeIndex to -1. This will be
    // used to store the menu index that corresponds to
    // the value of the TYPE attribute
    var typeIndex = -1;

    // If there was a TYPE attribute
    if (theType){
        // If the value of TYPE is "jeepers", set typeIndex to 0
        if (theType.toLowerCase() == "jeepers"){
            typeIndex = 0;
        }
        // If the value of TYPE is "jinkies", set typeIndex to 1
        }else if (theType.toLowerCase() == "jinkies"){
            typeIndex = 1;
        }
        // If the value of TYPE is "zoinks", set typeIndex to 2
        }else if (theType.toLowerCase() == "zoinks"){
            typeIndex = 2;
        }
    }
}

// If the value of the TYPE attribute was "jeepers",
// "jinkies", or "zoinks", choose the corresponding
// option from the pop-up menu in the interface
```

```

    if (typeIndex != -1){
        document.topLayer.document.topLayerForm.intType.selectedIndex = typeIndex;
    }
}

function setInterjectionTag(){
    // Get the DOM of the current document
    var theDOM = dw.getDocumentDOM();
    // Get the selected node
    var theObj = theDOM.getSelectedNode();

    // Get the index of the selected option in the pop-up menu
    // in the interface
    var typeIndex = document.topLayer.document.topLayerForm.intType.selectedIndex;
    // Get the value of the selected option in the pop-up menu
    // in the interface
    var theType =
document.topLayer.document.topLayerForm.intType.options[typeIndex].value;

    // Set the value of the TYPE attribute to theType
    theObj.setAttribute('type',theType);
}

</SCRIPT>
</HEAD>

<BODY>
<SPAN ID="image" STYLE="position:absolute; width:23px; height:17px; z-index:16; left:
3px; top: 2px">
<IMG SRC="interjection.gif" WIDTH="36" HEIGHT="36" NAME="interjectionImage"> </
SPAN>
<SPAN ID="label" STYLE="position:absolute; width:23px; height:17px; z-index:16; left:
44px; top: 5px">Interjection</SPAN>

<!-- If your form fields are in different layers, you must create a separate form inside each
layer and reference it as shown in the inspectSelection() and setInterjectionTag() functions
above. -->

```

```
<SPAN ID="topLayer" STYLE="position:absolute; z-index:1; left: 125px; top: 3px; width:
431px; height: 32px">
<FORM NAME="topLayerForm">
<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0">
<TR>
<TD VALIGN="baseline" ALIGN="right">Type: </TD>
<TD VALIGN="baseline" ALIGN="right">
<SELECT NAME="intType" STYLE="width:86" onChange="setInterjectionTag()">
<OPTION VALUE="jeepers">Jeepers</OPTION>
<OPTION VALUE="jinkies">Jinkies</OPTION>
<OPTION VALUE="zoinks">Zoinks</OPTION>
</SELECT>
</TD>
</TABLE>
</FORM>
</SPAN>

</BODY>
</HTML>
```



## CHAPTER 13

### Floating Palettes

.....

In Dreamweaver 3, you can now create any kind of floating palette or inspector without the size and layout limitations of property inspectors.

A custom property inspector should still be your first choice for setting the properties of the current selection. However, custom floating palettes offer more room and flexibility for displaying information about the entire document or multiple selections.

Dreamweaver already has several built-in floating palettes that are accessible from the Window menu; you can add your own palettes to this menu using the extensible menus feature. For more information on adding items to the menu system, see Chapter 16, “Customizing Dreamweaver,” in *Using Dreamweaver*.

Floating palette files are HTML files that reside in the Configuration/Floaters folder inside the Dreamweaver application folder. The BODY of a floating palette file contains an HTML form; event handlers attached to form elements may call JavaScript code that performs arbitrary edits to the current document.

### How floating palette files work

Custom floating palettes can be moved, resized, and tabbed together just like the floating palettes that are built into Dreamweaver. Custom floaters differ from built-in floaters in the following ways:

- It is not possible to display an icon in the tab of a custom floating palette; the tab always shows the contents of the floater's TITLE tag.
- Custom floating palettes display in the default gray. Setting the BGCOLOR attribute in the BODY tag has no effect.
- All custom floating palettes either appear always on top of the Document window or float behind it when inactive, depending on the setting for All Other Floaters in the Floating Palettes preferences.

Floating palette files also differ somewhat from other extensions. Unlike other extension files, Dreamweaver does not load floating palette files into memory at startup unless the floaters were visible when Dreamweaver last shut down. If the floaters were not visible when Dreamweaver last shut down, the files that define them are only loaded when referenced from one of the following functions:

`dreamweaver.getFloaterVisibility()`, `dreamweaver.setFloaterVisibility()`, or `dreamweaver.toggleFloater()`.

When one of the files inside the Configuration folder calls

`dw.getFloaterVisibility(floaterName)`, `dw.setFloaterVisibility(floaterName)`, or `dw.toggleFloater(floaterName)`, the following chain of events occurs:

- 1 If *floaterName* is not one of the reserved floater names, Dreamweaver searches the Configuration/Floaters folder for a file called *floaterName*.htm. (For a complete list of reserved floater names, see “`dreamweaver.getFloaterVisibility()`” on page 151.) If *floaterName*.htm is not found, Dreamweaver searches for *floaterName*.html. If no file is found, nothing further happens.
- 2 If the floating palette file is being loaded for the first time, the `initialPosition()` function is called, if defined, to determine the floater’s default position on the screen, and the `initialTabs()` function is called, if defined, to determine the floater’s default tab grouping.
- 3 The `selectionChanged()` and `documentEdited()` functions are called on the assumption that changes probably occurred while the floater was hidden.
- 4 When the floater is visible, the following things happen:
  - When the selection changes, the `selectionChanged()` function is called, if defined.
  - When the user makes changes to the document, the `documentEdited()` function is called, if defined.
  - Event handlers attached to the fields in the floater interface execute as the user encounters them. (For example, a button with an `onClick` event handler that calls `dw.getDocumentDOM().body.innerHTML=""` would remove everything between the opening and closing BODY tags in the document when clicked.)
- 5 When the user quits Dreamweaver, the current visibility, position, and tab grouping of the floater are saved. The next time Dreamweaver starts up, it loads the floating palette files for any floaters that were visible at the last shutdown and displays the floaters in their last position and tab grouping.

## The floating palette API

There are four custom functions in the floating palette API, and none is required. The functions in the floating palette API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of floating palette files. That is, Dreamweaver automatically calls the `documentEdited()` function if it is defined in a floating palette file, whereas in any other extension file a function named `documentEdited()` acts like a user-defined function—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions in the main API work: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

### `documentEdited()`

**Description** Called when the floater becomes visible and after the current series of edits is complete—that is, multiple edits may occur before this function is called. This function should be defined only if the floater must track edits to the document.

**Note:** Only define `documentEdited()` only if you absolutely require it, because its existence impacts performance.

**Arguments** None.

**Returns** Nothing.

**Example** The following instance of `documentEdited()` scans the document for layers and updates a text field that displays the number of layers in the document:

```
function documentEdited(){
  /* create a list of all the layers in the document */
  var theDOM = dw.getDocumentDOM();
  var layersInDoc = theDOM.getElementsByTagName("layer");
  var layerCount = layersInDoc.length;

  /* update the numOfLayers field with the new layer count */
  document.theForm.numOfLayers.value = layerCount;
}
```

## selectionChanged()

**Description** Called when the floater becomes visible, and whenever the selection changes (when focus switches to a new document, or when the insertion pointer moves to a new location in the current document). This function should be defined only if the floater must track the selection.

**Note:** Only define `selectionChanged()` if you absolutely require it, because its existence impacts performance.

**Arguments** None.

**Returns** Nothing.

**Example** The following instance of `selectionChanged()` shows a different panel (layer) in the floater depending on whether the selection is a script marker or something else:

```
function selectionChanged(){
    /* get the selected node */
    var theDOM = dw.getDocumentDOM();
    var theNode = dw.getSelectedNode();

    /* check to see if the node is a script marker */
    if (theNode.nodeType == Node.ELEMENT_NODE && theNode.tagName == "SCRIPT"){
        document.layers['blanklayer'].visibility = 'hidden';
        document.layers['scriptlayer'].visibility = 'visible';
    }else{
        document.layers['scriptlayer'].visibility = 'hidden';
        document.layers['blanklayer'].visibility = 'visible';
    }
}
```

## initialPosition()

**Description** Determines the initial position of the floater the first time it is called. If this function is not defined, the default position is the center of the screen.

**Arguments** *platform*

Possible values for *platform* are "Mac" and "Win".

**Returns** A string of the form "leftPosInPixels,topPosInPixels".

**Example** The following instance of `initialPosition()` specifies that the first time the floater appears, it should be 420 pixels from the left and 20 pixels from the top in Windows, and 400 pixels from the left side of the screen and 20 pixels from the top of the screen on the Macintosh:

```
function initialPosition(platform){
    var initPos = "420,20";
    if (platform == "macintosh"){
        initPos = "390,20";
    }
    return initPos;
}
```



## initialTabs()

<b>Description</b>	Determines which other floaters are tabbed together with this one the first time the floater appears. If any listed floater has appeared previously, it is not included in the tab group. Thus, to assure that two custom floaters are tabbed together, each should reference the other in each initialTabs() function.
<b>Arguments</b>	None.
<b>Returns</b>	A string of the form "floaterName1,floaterName2,...floaterNameN".
<b>Example</b>	<p>The following instance of initialTabs() specifies that the first time the floater appears, it should be tabbed together with the scriptEditor floater:</p> <pre>function initialTabs(){     return "scriptEditor"; }</pre>

## About performance

Declaring the selectionChanged() or documentEdited() function in your custom floaters, risks adversely impacting Dreamweaver's performance. Consider that documentEdited() is called after every keystroke, and selectionChanged() is called after every press of an arrow key. It's important to test your floater against many different scenarios, using large documents (100K or more of HTML) whenever possible.

To help you avoid performance penalties, setTimeout() has been implemented as a global method in Dreamweaver 3. As in the browsers, setTimeout() takes two arguments: the JavaScript to be called, and the amount of time in milliseconds to wait before calling it.

The setTimeout() method lets you build pauses into your processing during which the user can continue interacting with the application. You must build in these pauses explicitly because the screen freezes while scripts are processing, preventing the user from performing further edits (and you from updating the interface or the floater).

The following code is from a floater that displays information about every layer in the document. It uses `setTimeout()` to pause for half a second after processing each layer:

```
/* create a flag that specifies whether an edit is being processed, and set it to FALSE. */
document.running = false;

/* this function called when document is edited */
function documentEdited(){
    /* create a list of all the layers to be processed */
    var dom = dw.getDocumentDOM();
    document.layers = dom.getElementsByTagName("layer");
    document.numLayers = document.layers.length;
    document.numProcessed = 0;

    /* set a timer to call processLayer(); if we didn't get
    * to finish processing the previous edit, then the timer
    * is already set. */
    if (document.running = false){
        setTimeout("processLayer()", 500);
    }

    /* set the processing flag to TRUE */
    document.running = true;
}

/* process one layer */
function processLayer(){
    /* display information for the next unprocessed layer.
    displayLayer() is a function you would write to
    perform the "magic". */
    displayLayer(document.layers[document.numProcessed]);

    /* if there's more work to do, set a timeout to process
    * the next layer. If we're finished, set the document.running
    * flag to FALSE. */
    document.numProcessed = document.numProcessed + 1;
    if (document.numProcessed < document.numLayers){
        setTimeout("processLayer()", 500);
    }else{
        document.running = false;
    }
}
```

## A simple floating palette example

The following floating palette contains a text field that shows the contents of the selected Script marker (the yellow icon that appears in the Document window to mark the location of a script). If no Script marker is selected, a layer that contains the text (no script selected) appears.

```
<html>
<head>
<title>Script Editor</title>
<script language="JavaScript">

function selectionChanged(){
  /* get the selected node */
  var theDOM = dw.getDocumentDOM();
  var theNode = theDOM.getSelectedNode();

  /* check to see if the node is a script marker */
  if (theNode.nodeType == Node.ELEMENT_NODE && theNode.tagName == "SCRIPT"){
    document.layers['scriptlayer'].visibility = 'visible';
    document.layers['scriptlayer'].document.theForm.scriptCode.value = theNode.innerHTML;
    document.layers['blanklayer'].visibility = 'hidden';
  }else{
    document.layers['scriptlayer'].visibility = 'hidden';
    document.layers['blanklayer'].visibility = 'visible';
  }
}

/* update the document with any changes made by
the user in the textarea */
function updateScript(){
  var theDOM = dw.getDocumentDOM();
  var theNode = dw.getSelectedNode();
  theNode.innerHTML = document.layers['scriptlayer'].document.theForm.scriptCode.value;
}

</script>
</head>

<body>
<div id="blanklayer" style="position:absolute; width:422px; height:181px; z-index:1; left:
8px; top: 11px; visibility: hidden">
<center>
<br>
<br>
<br>
<br>
<br>
(no script selected)
</center>
</div>
```

```

<div id="scriptlayer" style="position:absolute; width:422px; height:181px; z-index:1; left:
8px; top: 11px; visibility: visible">
<form name="theForm">
<textarea name="scriptCode" cols="80" rows="20" wrap="VIRTUAL"
onBlur="updateScript()"></textarea>
</form>
</div>

</body>
</html>

```

Remember that it is not sufficient to save this code in a file called `scriptEditor.htm` in the `Configuration/Floater` folder; you must also call `dw.setFloaterVisibility('scriptEditor',true)`, or `dw.toggleFloater('scriptEditor')` from somewhere in order to load the floater and make it visible. The most obvious place from which to do this is the `Window` menu in `menus.xml` file. The `menuitem` tag might look like this:

```

<menuitem name="Script Editor" enabled="true"
command="dw.toggleFloater('scriptEditor')"
checked="dw.getFloaterVisibility('scriptEditor')"/>

```

## CHAPTER 14

### Behaviors

---

Behaviors let nonprogrammers make their HTML pages interactive. They offer web designers an easy way to assign an actions to page elements by filling in an HTML form.

You should write behavior actions when you want to share functions with nonprogrammers, or when you want to insert the same JavaScript function repeatedly but change the parameters each time.

**Note:** You cannot use behaviors to insert VBScript functions directly; however, you can add a VBScript function indirectly by editing the DOM in the `applyBehavior()` function.

The term “behavior” refers to the combination of an event (such as `onClick`, `onLoad`, or `onSubmit`) and an action (for example, Check Plugin, Go to URL, Swap Image). The browser determines which HTML elements accept which events. Files listing events that each browser supports are stored in the `Configuration/Behaviors/Events` folder within the Dreamweaver application folder.

Actions are HTML files. The `BODY` of an action file generally contains an HTML form that accepts parameters for the action (for example, parameters indicating which layers are to be shown or hidden). The `HEAD` of an action file contains JavaScript functions that process form input from the `BODY` and control the functions, arguments, and event handlers that are inserted into a user’s document.

## How behaviors work

When a user selects an HTML element in a Dreamweaver document and opens the Behavior inspector, the following chain of events occurs:

- 1 The user clicks the + button to display the Actions pop-up menu.
- 2 Dreamweaver calls the `canAcceptBehavior()` function in each action file to see whether this action is appropriate for the document or the selected element. If the return value of this function is `FALSE`, Dreamweaver dims the action in the Actions pop-up menu. (For example, the Control Shockwave action is dimmed when the user's document has no Shockwave movies.) If the return value is a list of events, Dreamweaver compares each event with the valid events for the currently selected HTML element and target browser until it finds a match.
- 3 Dreamweaver populates the Events pop-up menu with the matched event from `canAcceptBehavior()` at the top of the list; if no match exists, the default event for the HTML element (marked in the event file with an asterisk) becomes the top item. The remaining events in the menu are culled from the event file.
- 4 The user selects an action from the Action pop-up menu.
- 5 Dreamweaver calls the `windowDimensions()` function, if defined, to determine the size of the parameters dialog box. If `windowDimensions()` is not defined, the size is determined automatically.
- 6 Dreamweaver displays a dialog box containing the `BODY` elements of the action file. If the action file's `BODY` tag contains an `onLoad` handler, Dreamweaver executes it.
- 7 The user fills in the parameters for the action. Dreamweaver executes event handlers associated with the form as the user encounters them.
- 8 The user clicks OK.
- 9 Dreamweaver calls the `behaviorFunction()` and `applyBehavior()` functions in the selected action file. These functions return strings that are inserted into the user's document.
- 10 If the user later double-clicks the action in the Actions column, Dreamweaver reopens the parameters dialog box, executing the `onLoad` handler. Dreamweaver then calls the `inspectBehavior()` function in the selected action file, which fills in the fields with the data that the user entered previously.

## Inserting multiple functions in the user's file

Actions can insert multiple functions—the main behavior function plus any number of helper functions—into the HEAD. Two or more behaviors can even share helper functions, as long as the function definition is exactly the same in each action file. One way of ensuring that shared functions are identical is to store each helper function in an external JavaScript file and insert it into the appropriate action files using `<SCRIPT SRC="externalFile.js">`.

When the user deletes a behavior, Dreamweaver will attempt to remove any unused helper functions associated with the behavior. If other behaviors are using a helper function, it will not be deleted. Because the algorithm for deleting helper functions errs on the side of caution, Dreamweaver may occasionally leave behind an unused function in the user's document.

## What to do when an action requires a return value

Sometimes an event handler must have a return value (for example, `onMouseOver="window.status='This is a link'; return true'"`). But if Dreamweaver inserts `"return behaviorName(args)"` into the event handler, behaviors later in the list are skipped.

To get around this limitation, set a variable called `document.MM_returnValue` to the desired return value within the string returned by `behaviorFunction()`. This setting causes Dreamweaver to insert `return document.MM_returnValue` at the end of the list of actions in the event handler. See the `Validate Form.js` file in the `Configuration/Behaviors/Actions` folder within the Dreamweaver application folder for an example of the use of `MM_returnValue`.

## The behavior API

There are eight custom functions in the behavior API, two of which (`applyBehavior()` and `behaviorFunction()`) are required. The functions in the behavior API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of behavior files. That is, Dreamweaver automatically calls the `applyBehavior()` function if it is defined in a behavior file, whereas in any other extension file a function named `applyBehavior()` acts like a user-defined function—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions in the main API work: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

## applyBehavior()

**Description** Inserts into the user's document an event handler that calls the function inserted by behaviorFunction(). This function can also perform other edits on the user's document, but it must not delete the object to which the behavior is being applied or the object that receives the action.

**Arguments** *uniqueName*

The argument is a unique identifier among all instances of all behaviors in the user's document. Its format is *functionNameInteger*, where *functionName* is the name of the function inserted by behaviorFunction(). This argument is useful if you insert a tag into the user's document and you want to assign a unique value to its NAME attribute.

**Returns** A string containing the function call to be inserted in the user's document, usually after accepting parameters from the user. If applyBehavior() determines that the user made an invalid entry, the function can return an error string instead of the function call. If the string is empty (return "") reports no error; but if the string is non-empty and not a function call, Dreamweaver displays a dialog box with the text: Invalid Input supplied for this behavior: [the string returned from applyBehavior()]. If the return value is NULL (return;), Dreamweaver indicates that an error occurred but offers no specific information.

**Note:** Quotation marks within the returned string must be preceded by a backslash (\) to avoid errors reported by the JavaScript interpreter.

**Example** The following instance of applyBehavior() returns a call to the function MM\_openBrWindow() and passes it parameters given by the user (the height and width of the window; whether the window should have scroll bars, a toolbar, a location bar, and other features; and the URL that should open in the window):

```
function applyBehavior() {
    var i,theURL,theName,arrayIndex = 0;
    var argArray = new Array(); //use array to produce correct number of commas w/o spaces
    var checkBoxNames = new
    Array("toolbar","location","status","menubar","scrollbars","resizable");

    for (i=0; i<checkBoxNames.length; i++) {
        theCheckBox = eval("document.theForm." + checkBoxNames[i]);
        if (theCheckBox.checked) argArray[arrayIndex++] = (checkBoxNames[i] + "yes");
    }
    if (document.theForm.width.value)
        argArray[arrayIndex++] = ("width=" + document.theForm.width.value);
    if (document.theForm.height.value)
        argArray[arrayIndex++] = ("height=" + document.theForm.height.value);
    theURL = escape(document.theForm.URL.value);
    theName = document.theForm.winName.value;
    return "MM_openBrWindow('" + theURL + "','" + theName + "','" + argArray.join(',') + "')";
}
```



## behaviorFunction()

<b>Description</b>	Inserts one or more functions—surrounded by <code>&lt;SCRIPT LANGUAGE="JavaScript"&gt;</code> <code>&lt;/SCRIPT&gt;</code> tags, if none yet exist—into the HEAD of the user's document.
<b>Arguments</b>	None.
<b>Returns</b>	Either a string containing the JavaScript functions to be inserted in the user's document, or a string containing the names of the functions to be inserted into the user's document. This value must be exactly the same every time (it cannot depend on input from the user). The functions are inserted only once, regardless of how many times the action is applied to elements in the document.

**Note:** Quotation marks within the returned string must be preceded by a backslash (\) to avoid errors reported by the JavaScript interpreter.

**Example** The following instance of `behaviorFunction()` returns a function called `MM_popupMsg()`:

```
function behaviorFunction(){
    return "" +
    "function MM_popupMsg(theMsg) { //v1.0\n"+
    "  alert(theMsg);\n"+
    "};\n";
}
```

The following code would be equivalent to the preceding `behaviorFunction()` declaration, and is the method used to declare `behaviorFunction()` in all of the behaviors that ship with Dreamweaver:

```
function MM_popupMsg(theMsg){ //v1.0
    alert(theMsg);
}

function behaviorFunction(){
    return "MM_popupMsg";
}
```

## canAcceptBehavior()

**Description** Determines whether the action is allowed for the selected HTML element and specifies the default event that should trigger the action. May also check for the existence of certain objects (such as Shockwave movies) in the user's document and disallow the action if these objects do not appear.

**Arguments** *HTMLLelement*

The argument is the selected HTML element.

**Returns** One of the following values:

- TRUE if the action is allowed but has no preferred events.
- A list of preferred events (in descending order of preference) for this action. Specifying preferred events overrides the default event (as denoted with an asterisk in the event file) for the selected object. See steps 2 and 3 in "How behaviors work" on page 262.
- FALSE if the action is not allowed.

If `canAcceptBehavior()` returns FALSE, the action is dimmed in the Actions pop-up menu in the Behavior inspector.

**Example** The following instance of `canAcceptBehavior()` returns a list of preferred events for the behavior if the document has any named images:

```
function canAcceptBehavior(){
    var theDOM = dreamweaver.getDocumentDOM();
    // Get an array of all images in the document
    var allImages = theDOM.getElementsByTagName('IMG');
    if (allImages.length > 0){
        return "onMouseOver, onClick, onMouseDown";
    }else{
        return FALSE;
    }
}
```

## displayHelp()

**Description** If this function is defined, a Help button appears below the OK and Cancel buttons in the parameters dialog box. This function is called when the user clicks the Help button.

**Arguments** None.

**Returns** Nothing.

**Example** The following instance of `displayHelp()` opens in a browser window a file with instructions for using the behavior:

```
function displayHelp(){
    dreamweaver.browseDocument('http://www.hotwired.com/webmonkey/javascript/
code_library/wm_pos2_elmnt_dw/?tw=javascript');
}
```

## deleteBehavior()

<b>Description</b>	<p>Undoes any edits performed by <code>applyBehavior()</code>.</p> <p><b>Note:</b> Dreamweaver automatically deletes the function declaration and the event handler associated with a behavior when the user deletes the behavior in the Behavior inspector. Thus, it is necessary to define <code>deleteBehavior()</code> only if the <code>applyBehavior()</code> function performed additional edits on the user's document (for example, if it inserted an <code>EMBED</code> tag).</p>
<b>Arguments</b>	<p><i>applyBehaviorString</i></p> <p>This argument is the string that was returned earlier by the <code>applyBehavior()</code> function.</p>
<b>Returns</b>	Nothing.

## identifyBehaviorArguments()

<b>Description</b>	<p>Identifies arguments from a behavior function call as <code>nav</code>, <code>dep</code>, <code>URL</code>, <code>NS4.0ref</code>, <code>IE4.0ref</code>, <code>objName</code>, or other so that URLs in behaviors can be updated if the user saves the document to another location, and so that the referenced files can be displayed in the site map and considered dependent files for the purposes of uploading to and downloading from a server.</p>
<b>Arguments</b>	<p><i>theFunctionCall</i></p> <p>This argument is the string that was returned earlier by the <code>applyBehavior()</code> function.</p>
<b>Returns</b>	<p>A string containing a comma-separated list of the types of arguments in the function call. The length of the list must equal the number of arguments in the function call. Argument types will always be one of the following:</p> <ul style="list-style-type: none"><li>• <code>nav</code> specifies that the argument is a navigational URL, and therefore that it should be displayed in the site map.</li><li>• <code>dep</code> specifies that the argument is a dependent file URL, and therefore that it should be included with all other dependent files when a document containing this behavior is downloaded from or uploaded to a server.</li><li>• <code>URL</code> specifies that the argument is both a navigational URL and a dependent URL (or that it is a URL of unknown type), and therefore that it should be displayed in the site map and considered a dependent file when uploading to or downloading from a server.</li><li>• <code>NS4.0ref</code> specifies that the argument is a Netscape DOM object reference.</li><li>• <code>IE4.0ref</code> specifies that the argument is an Internet Explorer DOM object reference.</li><li>• <code>objName</code> specifies that the argument is a simple object name, as specified in the <code>NAME</code> attribute for the object. This type was added in Dreamweaver 3.</li><li>• <code>other</code> specifies that the argument is none of the above types.</li></ul>

**Example** This simple example of `identifyBehaviorArguments()` would work for the Open Browser Window behavior action, which returns a function that always has three arguments (the URL to open, the name of the new window, and the list of window properties):

```
function identifyBehaviorArguments(fnCallStr) {  
    return "URL,other,other";  
}
```

A more complex version of `identifyBehaviorArguments()` is necessary for behavior functions that have a variable number of arguments (such as Show/Hide Layer). This version of `identifyBehaviorArguments()` relies on the fact that there is a minimum number of arguments, and additional arguments always come in multiples of the minimum number. In other words, a function with a minimum number of arguments of 4 may have 4, 8, or 12 arguments, but it will never have 10 arguments.

```
function identifyBehaviorArguments(fnCallStr) {  
    var listOfArgTypes;  
    var itemArray = dreamweaver.getTokens(fnCallStr, '(),,');  
  
    //The array of items returned by getTokens() includes the function name,  
    //so the number of *arguments* in the array is the length of the array  
    //minus one. Divide by 4 to get the number of groups of arguments.  
    var numArgGroups = ((itemArray.length - 1)/4);  
    //For each group of arguments  
    for (i=0; i < numArgGroups; i++){  
  
        //Add a comma and "NS4.0ref,IE4.0ref,other,dep" (because this  
        //hypothetical behavior function has a minimum of four arguments:  
        //the Netscape object reference, the IE object reference, a dependent  
        //URL, and perhaps a property value such as "show" or "hide") to the  
        //existing list of argument types, or if no list yet exists, add only  
        //"NS4.0ref,IE4.0ref,other,dep"  
        var listOfArgTypes += ((listOfArgTypes)?",":"" ) +  
            "NS4.0ref,IE4.0ref,other,dep";  
    }  
}
```

## inspectBehavior()

**Description** Inspects the function call for a previously applied behavior in the user's document and sets the values of the options in the parameters dialog box accordingly. If inspectBehavior() is not defined, the default option values appear.

**Note:** inspectBehavior() must rely solely on information passed to it via the *applyBehaviorString* argument. Do not attempt to obtain other information about the user's document (for example, using `dreamweaver.getDocumentDOM()`) within this function.

**Arguments** *applyBehaviorString*

This argument is the string that was returned earlier by the `applyBehavior()` function.

**Returns** Nothing.

**Example** The following instance of `inspectBehavior()`, taken from `Display Status Message.htm`, fills in the Message field in the parameters form with the message that the user selected when the behavior was originally applied:

```
function inspectBehavior(msgStr){
    var startStr = msgStr.indexOf("") + 1;
    var endStr = msgStr.lastIndexOf("");
    if (startStr > 0 && endStr > startStr) {
        document.theForm.message.value = unescQuotes(msgStr.substring(startStr,endStr));
    }
}
```

**Note:** For more information about the `unescQuotes()` function, see the `string.js` file in the `Configuration/Shared/macromedia/scripts/cmn` folder.

## windowDimensions()

<b>Description</b>	Sets specific dimensions for the parameters dialog box to speed up display. If this function is not defined, the window dimensions are computed automatically.  <b>Note:</b> Do not define this function unless you want an options dialog box larger than 640 pixels by 480 pixels.
<b>Arguments</b>	<i>platform</i>  The value of the argument is either "macintosh" or "windows", depending on the user's platform.
<b>Returns</b>	A string of the form "widthInPixels,heightInPixels".  The returned dimensions are smaller than the size of the entire dialog window because they do not include the area for the OK and Cancel buttons. If the returned dimensions do not accommodate all options, scroll bars appear.
<b>Example</b>	The following instance of windowDimensions() sets the dimensions of the parameters dialog box to 648 pixels by 520 pixels:  <pre>function windowDimensions(platform){     return "648,520"; }</pre>

## A simple behavior example

To understand better how behaviors work and how you can create one, it's helpful to look at an example. The Configuration/Behaviors/Actions folder inside the Dreamweaver application folder is full of examples; however, many of these are likely to be too complex a starting point for all but the most advanced developers. The simplest action file to start with is Call JavaScript.htm (along with its counterpart, Call JavaScript.js, which contains all the JavaScript functions).

The following code also presents a relatively simple example. It checks the brand of the browser and goes to one page if the brand is Netscape, and another if the brand is Microsoft Internet Explorer. This code could easily be expanded to check for other brands—such as Opera and WebTV—and modified to perform other actions besides going to URLs.

```
<html>
<head>
<title>behavior "Check Browser Brand"</title>
<meta http-equiv="Content-Type" content="text/html">
<script language="JavaScript">

// The function that will be inserted into the
// HEAD of the user's document
function checkBrowserBrand(netscapeURL,explorerURL) {
  if (navigator.appName == "Netscape") {
    if (netscapeURL) location.href = netscapeURL;
  }else if (navigator.appName == "Microsoft Internet Explorer") {
    if (explorerURL) location.href = explorerURL;
  }
}

//***** API *****/

function canAcceptBehavior(){
  return true;
}

// Return the name of the function to be inserted into
// the HEAD of the user's document
function behaviorFunction(){
  return "checkBrowserBrand";
}

// Create the function call that will be inserted
// with the event handler
function applyBehavior() {
  var nsURL = escape(document.theForm.nsURL.value);
  var ieURL = escape(document.theForm.ieURL.value);
  if (nsURL && ieURL) {
    return "checkBrowserBrand(\" + nsURL + "\",\" + ieURL + "\")";
  }else{
    return "Please enter URLs in both fields."
  }
}

// Extract the arguments from the function call
// in the event handler and repopulate the
// parameters form
function inspectBehavior(fnCall){
  var argArray = getTokens(fnCall, "(");
  var nsURL = unescape(argArray[1]);
```

```

var ieURL = unescape(argArray[2]);
document.theForm.nsURL.value = nsURL;
document.theForm.ieURL.value = ieURL;
}

//***** LOCAL FUNCTIONS *****

// Put the cursor in the first text field
// and select the contents, if any
function initializeUI(){
    document.theForm.nsURL.focus();
    document.theForm.nsURL.select();
}

// Let the user browse to the Navigator and
// IE URLs
function browseForURLs(whichButton){
    var theURL = dreamweaver.browseForFileURL();
    if (whichButton == "nsURL"){
        document.theForm.nsURL.value = theURL;
    }else{
        document.theForm.ieURL.value = theURL;
    }
}

//***** END OF JAVASCRIPT *****
</script>
</head>
<body>
<form method="post" action="" name="theForm">
<table border="0" cellpadding="8">
<tr>
<td nowrap="nowrap">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&Go to this URL if the browser is Netscape
Navigator:<br>
<input type="text" name="nsURL" size="50" value=""> &nbsp;  &
<input type="button" name="nsBrowse" value="Browse..."
onClick="browseForURLs('nsURL')"></td>
</tr>
<tr>
<td nowrap="nowrap">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&Go to this URL is the browser is Microsoft Internet
Explorer:<br>
<input type="text" name="ieURL" size="50" value=""> &nbsp;  &
<input type="button" name="ieBrowse" value="Browse..."
onClick="browseForURLs('ieURL')"></td>
</tr>
</table>
</form>
</body>
</html>

```



## CHAPTER 15

### Data Translators

.....

Data translators translate specialized markup—such as server-side includes, conditional JavaScript statements, or other non-HTML code such as ePerl, PHP3, JSP, CFML, or ASP—into HTML code that can be read and displayed by Dreamweaver. In Dreamweaver 3 you can translate attributes within tags as well as entire tags or blocks of code.

Translated tags or blocks of code must be enclosed in locked regions to preserve the original markup. Translated attributes do not require locks, which makes inspecting the tags that contain them a simpler process.

All data translators—block/tag or attribute—are HTML files. The HEAD of a data translator contains two or more JavaScript functions: one that specifies what the translator does and the files it acts on, one that performs the actual translation, and any number of supporting functions. The BODY of a data translator is empty.

Data translation—especially for entire tags or blocks of code—might involve complex operations that either cannot be done with JavaScript or that would be more efficiently done with C. If you are familiar with C or C++, you should also read “C-Level Extensibility” on page 209.

## How data translators work

Dreamweaver handles all translator files the same way, regardless of whether they translate entire tags or only attributes. At startup, Dreamweaver reads all the files in the Configuration/Translators folder and calls the `getTranslatorInfo()` function to obtain information about the translator. Dreamweaver ignores any file in which `getTranslatorInfo()` does not exist or contains an error that causes it to be undefined.

**Note:** To prevent JavaScript errors from interfering with startup, errors in any translator file are reported only after all translators are loaded. For more information on debugging translators, see “Finding bugs in your translator” on page 293.

Dreamweaver also calls the `translateMarkup()` function in all applicable translator files (as specified in the Translation preferences) whenever the user may have added new—or changed existing—content that needs translation. Dreamweaver therefore calls `translateMarkup()` when the user:

- Opens a file in Dreamweaver.
- Switches back to the Document window after making changes in the HTML inspector.
- Changes the properties of an object in the current document.
- Inserts an object (using either the Object palette or the Insert menu).
- Refreshes the current document after making changes to it in another application.
- Applies a template to the document.
- Pastes or drags content into or within the Document window.
- Saves changes to a dependent file.
- Invokes a command, behavior, property inspector, or other extension that sets the `innerHTML` or `outerHTML` property of any tag object or the `data` property of any comment object.
- Chooses File > Convert > 3.0 Browser Compatible.
- Chooses Modify > Layout Mode > Convert Tables to Layers.
- Chooses Modify > Layout Mode > Convert Layers to Tables.
- Chooses Modify > Translate > *translatorName*.
- Changes a tag or attribute in the Quick Tag Editor and presses Tab or Enter.

## The data translator API

There are only two custom functions in the data translator API, but both are required. The functions in the data translator API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of data translator files. That is, Dreamweaver automatically calls the `translateMarkup()` function if it is defined in a data translator file, whereas in any other extension file a function named `translateMarkup()` acts like a user-defined function—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions in the main API work: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

### `getTranslatorInfo()`

<b>Description</b>	Provides information about the translator and the files it can act upon.
<b>Arguments</b>	None.
<b>Returns</b>	<p>An array of strings. The elements of the array must appear in the following order:</p> <ul style="list-style-type: none"><li>• <i>translatorClass</i> uniquely identifies the translator. This string must begin with a letter and can contain only alphanumeric characters, hyphens (-), and underscores (_).</li><li>• <i>title</i> describes the translator in no more than 40 characters. This string appears in the Modify &gt; Translate menu as well as in the Translation preferences.</li><li>• <i>nExtensions</i> specifies the number of file extensions to follow. If <i>nExtensions</i> is 0, the translator can run on any file.</li><li>• <i>extension</i> specifies a file extension (for example, ".htm" or ".SHTML") that this translator can work with. This string is case insensitive and should not contain a leading period. The array should contain the same number of <i>extension</i> elements as are specified in <i>nExtensions</i>.</li><li>• <i>nRegExps</i> specifies the number of regular expressions that will follow. If <i>nRegExps</i> is 0, <i>runDefault</i> is the next element in the array.</li></ul>

- *regExps* specifies a regular expression to check for. The array should contain the same number of *regExps* elements as are specified in *nRegExps*, and at least one of the *regExps* must match a piece of the document's HTML source before the translator can act on a file.
- *runDefault* specifies the default preference for running this translator. Possible values are "allFiles", "noFiles", "byExtension", and "byExpression". If you set *runDefault* to "byExtension" but do not specify any extensions (see *extension*, above), the effect is the same as setting "allFiles". If you set *runDefault* to "byExpression" but do not specify any expressions (see *regExps*, above), the effect is the same as setting "noFiles". Whatever value you set for *runDefault* can be overridden by the user in the Preferences dialog box.

**Example** The following instance of `getTranslatorInfo()` gives information about a translator for server-side includes:

```
function getTranslatorInfo(){
    var transArray = new Array(11);

    transArray[0] = "SSI";
    transArray[1] = "Server-Side Includes";
    transArray[2] = "4";
    transArray[3] = "htm";
    transArray[4] = "stm";
    transArray[5] = "html";
    transArray[6] = "shtml";
    transArray[7] = "2";
    transArray[8] = "<!--#include file";
    transArray[9] = "<!--#include virtual";
    transArray[10] = "byExtension";

    return transArray;
}
```

## translateMarkup()

<b>Description</b>	Performs the translation.
<b>Arguments</b>	<i>docName, siteRoot, docContent</i> <ul style="list-style-type: none"><li>• The first argument is a string containing the file:// URL for the document to be translated.</li><li>• The second argument is a string containing the file:// URL for the root of the site that contains the document to be translated. If the document is outside a site, this string might be empty.</li><li>• The third argument is a string containing the contents of the document.</li></ul>
<b>Returns</b>	A string containing the translated document.
<b>Example</b>	The following instance of translateMarkup() calls the C function translateASP(), which is contained in a DLL (Windows) or code library (Macintosh) called ASPTrans:

```
function translateMarkup(docName, siteRoot, docContent){
    var translatedString = "";
    if (docContent.length > 0){
        translatedString = ASPTrans.translateASP(docName, siteRoot, docContent);
    }
    return translatedString;
}
```

For an all-JavaScript example, see “A simple attribute translator example” on page 279 or “A simple block/tag translator example” on page 285.

## Determining what kind of translator to use

All translators are the same to a certain extent: they must contain the getTranslatorInfo() and translateMarkup() functions, and they must reside in the Configuration/Translators folder. They differ, however, in the kind of code that they insert into the user’s document, and in how that code must be inspected.

- To translate small pieces of server markup that determine attribute values or that conditionally add attributes to a standard HTML tag, write an attribute translator. Standard HTML tags that contain translated attributes can be inspected with the property inspectors that are built into Dreamweaver. It is not necessary to write a custom property inspector. See “Adding a translated attribute to a tag” on page 278.
- To translate an entire tag (for example, a server-side include) or a block of code (for example, JavaScript, Cold Fusion, PHP, or other scripting), write a block/tag translator. The code generated by a block/tag translator cannot be inspected with the property inspectors that are built into Dreamweaver. You must write a custom inspector for the translated content if you want users to be able to change the properties of the original code. See “Locking translated tags or blocks of code” on page 284.

## Adding a translated attribute to a tag

Attribute translation relies heavily on the ability of the parser in Dreamweaver 3 to ignore server markup. Dreamweaver 3 already ignores the most common kinds of server markup (including ASP, CFML, and PHP) by default; if you are using server markup that has different start and end markers, you must modify the third-party tag database to ensure that your translator works properly. For more information on modifying the third-party tag database, see Chapter 16, “Customizing Dreamweaver,” in *Using Dreamweaver*.

With Dreamweaver handling the preservation of the original server markup, the translator’s task is to generate a valid attribute value that can be viewed in the Document window. (Hence, if you are using server markup only for attributes that do not have a user-visible effect, you do not need a translator.)

The translator creates an attribute value that has a visible effect in the Document window by adding a special attribute, `mmTranslatedValue`, to the tag containing the server markup. The `mmTranslatedValue` attribute and its value are not visible in the HTML inspector, however, nor are they saved with the document.

The `mmTranslatedValue` attribute must be unique within the tag. If it is likely that your translator will need to translate more than one attribute in a single tag, you must add a routine in the translator that appends numbers to `mmTranslatedValue` (for example, `mmTranslatedValue1`, `mmTranslatedValue2`, and so on).

The value of the `mmTranslatedValue` attribute must be a URL-encoded string containing at least one valid attribute/value pair. This means that `mmTranslatedValue="src=%22open.jpg%22"` is a valid translation for both `src="<? if (dayType == weekday) then open.jpg else closed.jpg" ?>` and `<? if (dayType == weekday) then src="open.jpg" else src="closed.jpg" ?>`. `mmTranslatedValue="%22open.jpg%22"` is not valid for either example because it contains only the value, not the attribute.

## Translating more than one attribute at a time

The mmTranslatedValue can contain more than one valid attribute/value pair. Consider the following untranslated code:

```
 alt="We're open 24 hours a day from  
12:01am Monday until 11:59pm Friday">
```

The translated markup might look like this:

```
  
mmTranslatedValue="src=%22open.jpg%22 width=%22320%22 height=%22100%22"  
alt="We're open 24 hours a day from 12:01am Monday until 11:59pm Friday">
```

Notice that the spaces between the attribute/value pairs in the mmTranslatedValue are not encoded. Because Dreamweaver looks for these spaces when it attempts to render the translated value, each attribute/value pair in the mmTranslatedValue must be encoded separately and then pieced back together to form the full mmTranslatedValue. For an example of how to do this, see “A simple attribute translator example” on page 279.

## A simple attribute translator example

To understand attribute translation a little better, it's helpful to look at an example. The following translator handles “Pound Conditional” (Poco) markup, a made-up syntax that's somewhat similar to ASP or PHP. The first step in making this translator work properly is to create a tagspec for Poco markup; this will prevent Dreamweaver from parsing the untranslated Poco statements.

The tagspec for Poco markup looks like this:

```
<tagspec tag_name="poco" start_string="<#" end_string="#>"  
detect_in_attribute="true" icon="poco.gif" icon_width="17"  
icon_height="15"></tagspec>
```

The poco.xml file containing this tagspec is stored in the Configuration/ThirdPartyTags folder along with the icon for Poco tags.

```
<html>
<head>
<title>Conditional Translator</title>
<meta http-equiv="Content-Type" content="text/html; charset=">
<script language="JavaScript">

/*****
 * This translator handles the following statement syntaxes:
 * <# if (condition) then foo else bar #>
 * <# if (condition) then att="foo" else att="bar" #>
 * <# if (condition) then att1="foo" att2="jinkies"
 * att3="jeepers" else att1="bar" att2="zoinks" #>
 *
 * It does not handle statements with no else clause.
 *****/

var count = 1;

function translateMarkup(docNameStr, siteRootStr, inStr){
  var count = 1;      // Counter to ensure unique mmTranslatedValues
  var outStr = inStr; // String that will be manipulated
  var spacer = "";    // String to manage space between encoded attributes
  var start = inStr.indexOf('<# if'); // First instance of Pound Conditional code

  /* Declared but not initialized. */
  var attAndValue;    // Boolean indicating whether the attribute is part of
                    // the conditional statement
  var trueStart;      // The beginning of the true case
  var falseStart;     // The beginning of the false case
  var trueValue;      // The HTML that would render in the true case
  var attName;        // The name of the attribute that is being'
                    // set conditionally.
  var equalSign;      // The position of the equal sign just to the
                    // left of the <#, if there is one
  var transAtt;       // The entire translated attribute
  var transValue;     // The value that must be URL-encoded
  var back3FromStart; // Three characters back from the start position
                    // (used to find equal sign to the left of <#
  var tokens;         // An array of all the attributes set in the true case
  var end;            // The end of the current conditional statement.

  // As long as there's still a <# conditional that hasn't been translated
  while (start != -1){
    back3FromStart = start-3;
    end = outStr.indexOf(' #>',start);
    equalSign = outStr.indexOf("=<# if",back3FromStart);
    attAndValue = (equalSign != -1)?false:true;
    trueStart = outStr.indexOf('then', start);
```



```

falseStart = outStr.indexOf(' else', start);
trueValue = outStr.substring(trueStart+5, falseStart);
tokens = dreamweaver.getTokens(trueValue,');

// If attAndValue is false, find out what attribute you're
// translating by backing up from the equal sign to the
// first space. The substring between the space and the
// equal sign is the attribute.
if (!attAndValue){
    for (var i=equalSign; i > 0; i--){
        if (outStr.charAt(i) == " "){
            attName = outStr.substring(i+1,equalSign);
            break;
        }
    }
    transValue = attName + '=' + trueValue + '';
    transAtt = ' mmTranslatedValue' + count + '=' + escape(transValue) + '';
    outStr = outStr.substring(0,end+4) + transAtt + outStr.substring(end+4);

// If attAndValue is true, and tokens is greater than
// 1, then trueValue is a series of attribute/value
// pairs, not just one. In that case, each attribute/value
// pair must be encoded separately and then added back
// together to make the translated value.
}else if (tokens.length > 1){
    transAtt = ' mmTranslatedValue' + count + '='
    for (var j=0; j < tokens.length; j++){
        tokens[j] = escape(tokens[j]);
        if (j>0){
            spacer=" ";
        }
        transAtt += spacer + tokens[j];
    }
    transAtt += '';
    outStr = outStr.substring(0,end+3) + transAtt + outStr.substring(end+3)

// If attAndValue is true and tokens is not greater
// than 1, then trueValue is a single attribute/value pair.
// This is the simplest case, where all that is necessary is
// to encode trueValue.
}else{
    transValue = trueValue;
    transAtt = ' mmTranslatedValue' + count + '=' + escape(transValue) + '';
    outStr = outStr.substring(0,end+3) + transAtt + outStr.substring(end+3);
}

// Increment the counter so that the next instance
// of mmTranslatedValue will have a unique name, and
// then find the next <# conditional in the code.
count++;

```

```

        start = outStr.indexOf('<# if,end);

    }

    // Return the translated string.
    return outStr
}

function getTranslatorInfo(){
    returnArray = new Array(7);

    returnArray[0] = "Pound_Conditional"; //The translatorClass
    returnArray[1] = "Pound Conditional Translator"; //The title
    returnArray[2] = "2"; //The number of extensions
    returnArray[3] = "html"; //The first extension
    returnArray[4] = "htm"; //The second extension
    returnArray[5] = "1"; // The number of expressions
    returnArray[6] = "<#"; //The first expression

    return returnArray
}

</script>
</head>

<body>
</body>
</html>

```

## Inspecting translated attributes

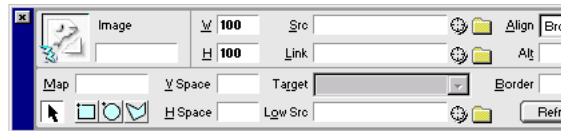
When server markup is used to specify a single attribute, and the attribute is represented in a Property inspector, Dreamweaver displays the server markup in the Property inspector.



The markup appears whether or not a translator is associated with it. If a translator is associated with the markup, a lightning bolt appears over the icon in the inspector. The translator runs whenever the user edits the server markup shown in the inspector.

**Note:** The lightning bolt icon does not appear when text or table cells, rows, or columns are selected. Translation still occurs if the user edits server markup in the inspector and a translator exists that handles that type of markup.

When server markup is used to control more than one attribute in a tag, the server markup does not show up in the Property inspector. However, the lightning bolt signals the user that translated markup exists for the selected element.



The fields in the inspector are still editable; users can enter values for attributes that may be controlled by server markup, resulting in duplicate attributes. If both a translated value and a regular value are set for a particular attribute, Dreamweaver displays the translated value in the Document window. You must decide whether your translator will look for duplicate attributes and remove them.

## Locking translated tags or blocks of code

In most cases, you want a translator to change markup so that Dreamweaver can display it, but you want the original markup—not the changes—to be saved. To address this need, Dreamweaver provides special XML tags in which to wrap translated content and to refer to the original code. The syntax of the XML tags is as follows:

```
<MM:BeginLock translatorClass="translatorClass" type="tagNameOrType"
depFiles="dependentFilesList" orig="encodedOriginalMarkup">
Translated content
<MM:EndLock>
```

where:

*translatorClass* is the unique identifier for the translator (the first string in the array returned by `getTranslatorInfo()`).

*tagNameOrType* is a string identifying the type of markup (or the tag name associated with the markup) contained in the lock. The string can contain only alphanumeric, hyphen (-), or underscore (\_) characters. You can check this value in the `canInspectSelection()` function of a custom property inspector to determine if the inspector is the right one for the content. For more information, see “Creating property inspectors for locked content” on page 289. Locked content cannot be inspected by any of Dreamweaver’s built-in property inspectors. For example, specifying `type="IMG"` does not make the Image inspector appear.

*dependentFilesList* is a string containing a comma-separated list of files on which the locked markup depends. Files are referenced from the hard drive (for example, `C:\sites\avocado8\copyright.html` in Windows or `MyHD:sites:avocado8:copyright.html` on Macintosh). If the user updates one of the files in the *dependentFilesList*, Dreamweaver automatically retranslates the content in the document containing the list.

*encodedOriginalMarkup* is a string containing the original, untranslated markup, encoded using a small subset of URL encoding (use %22 for “, %3C for <, %3E for >, and %25 for %). The quickest way to URL encode a string is to use the `escape()` method. For example, if `myString` equals `''`, `escape(myString)` returns `%3Cimg%20src=%22foo.gif%22%3E`.

The following example shows the locked portion of code that might be generated from the translation of the server-side include `<!--#include virtual="/footer.html" -->`:

```
<MM:BeginLock translatorClass="MM_SSI" type="ssi"
depFiles="C:\sites\webdev\footer.html" orig="%3C!--#include virtual=%22/
footer.html%22%20--%3E">
<!-- begin footer -->
<CENTER>
<HR SIZE=1 NOSHADE WIDTH=100%>

<BR>

[<A TARGET="_top" HREF="/">home</A>]
[<A TARGET="_top" HREF="/products/">products</A>]
[<A TARGET="_top" HREF="/services/">services</A>]
[<A TARGET="_top" HREF="/support/">support</A>]
[<A TARGET="_top" HREF="/company/">about us</A>]
[<A TARGET="_top" HREF="/help/">help</A>]
</CENTER>
<!-- end footer -->
<MM:EndLock>
```

### A simple block/tag translator example

To understand translation a little better, it's helpful to look at a translator that is written entirely in JavaScript (that is, one that does not rely on a C library for any functionality). The following translator would be more efficient if written in C, but the JavaScript version is simpler—which makes it perfect for demonstrating how translators work.

Like most translators, this one is designed to mimic server behavior. Assume that your web server is configured to replace the KENT tag with a different picture of an engineer depending on the day of the week, the time of day, and the user's platform. The translator does the same thing, only locally.

```
<html>
<head>
<title>Kent Tag Translator</title>
<meta http-equiv="Content-Type" content="text/html; charset=">
<script language="JavaScript">
/*****
 * The getTranslatorInfo() function provides information about the *
 * translator, including its class and name, the types of documents *
 * that are likely to contain the markup to be translated, the *
 * regular expressions that a document containing the markup to *
 * be translated would match, and the default Translation *
 * preference (whether the translator should run on all files, no *
 * files, in files with the specified extensions, or in files matching *
 * the specified expressions). *
 *****/
function getTranslatorInfo(){
    //Create a new array with 6 slots in it
    returnArray = new Array(6);

    returnArray[0] = "DREAMWEAVER_TEAM"    // The translatorClass
    returnArray[1] = "Kent Tags"           // The title
    returnArray[2] = "0"                   // The number of extensions
    returnArray[3] = "1"                   // The number of expressions
    returnArray[4] = "<kent"                 // Expression
    returnArray[5] = "byExpression"        // Default Translation preference
    return returnArray;
}

/*****
 * The translateMarkup() function performs the actual translation. *
 * In this translator, the translateMarkup() function is written *
 * entirely in JavaScript (that is, it does not rely on a C library) -- *
 * and it's also extremely inefficient. It's a simple example, however, *
 * which is good for learning. *
 *****/
function translateMarkup(docNameStr, siteRootStr, inStr){
    var outStr = "";                      // The string to be returned after translation
    var start = inStr.indexOf('<kent>');    // The first position of the KENT tag
                                          // in the document.

    var replCode = replaceKentTag();       // Calls the replaceKentTag() function
                                          // to get the code that will replace KENT.

    //If the document does not contain any content, terminate the translation.
    if ( inStr.length <= 0 ){
        return "";
    }
}
```

```

// As long as start, which is equal to the location in inStr of the
// KENT tag, is not equal to -1 (that is, as long as there is another
// KENT tag in the document)
while (start != -1){
    // Copy everything up to the start of the KENT tag.
    // This is very important, as translators should never change
    // anything other than the markup that is to be translated.
    outStr = inStr.substring(0, start);

    // Replace the KENT tag with the translated HTML, wrapped in special
    // locking tags. For more information on the replacement operation, see
    // the comments in the replaceKentTag() function.
    outStr = outStr + replCode;

    // Copy everything after the KENT tag.
    outStr = outStr + inStr.substring(start+6);

    // Use the string you just created for the next trip through
    // the document. This is the most inefficient part of all.
    inStr = outStr;
    start = inStr.indexOf('<kent>');
}
// When there are no more KENT tags in the document, return outStr.
return outStr;
}

/*****
* The replaceKentTag() function assembles the HTML that will
* replace the KENT tag and the special locking tags that will
* surround the HTML. It calls the getImage() function to
* determine the SRC of the IMG tag.
*****/
function replaceKentTag(){
    // The image to display.
    var image = getImage();
    // The location of the image on the local disk.
    var depFiles = dreamweaver.getSiteRoot() + image;
    // The IMG tag that will be inserted between the lock tags.
    var imgTag = '<IMG SRC="' + image + '" WIDTH="320" HEIGHT="240" ALT="Kent">\n';
    // The first part of the opening lock tag. The remainder of the tag is assembled below.
    var start = '<MM:BeginLock translatorClass="DREAMWEAVER_TEAM" type="kent"';
    // The closing lock tag.
    var end = '<MM:EndLock>';

    //Assemble the lock tags and the replacement HTML.
    var replCode = start + ' depFiles="' + depFiles + '"';
    replCode = replCode + ' orig="%3Ckent%3E">\n';
    replCode = replCode + imgTag;
    replCode = replCode + end;

    return replCode;
}

```

```

}

/*****
 * The getImage() function determines which image to display
 * based on the day of the week, the time of day and the
 * user's platform. The day and time are figured based on UTC
 * time (Greenwich Mean Time) minus 8 hours, which gives
 * Pacific Standard Time (PST). No allowance is made for Daylight
 * Savings Time in this routine.
 *****/
function getImage(){
    var today = new Date();           // Today's date & time.
    var day = today.getUTCDay();      // The day of the week in the GMT time zone.
                                        // 0=Sunday, 1=Monday, and so on.

    var hour = today.getUTCHours();   // The current hour in GMT, based on the
                                        // 24-hour clock.

    var SFhour = hour - 8;             // The time in San Francisco, based on the
                                        // 24-hour clock.

    var platform = navigator.platform; // The user's platform. All Windows machines
                                        // are identified by Dreamweaver as "Win32",
                                        // all Macs as "MacPPC".

    var imageRef;                     // The image reference to be returned.
    // If SFhour is negative, you have two adjustments to make.
    // First, you have to subtract one from the day count because it is already the wee
    // hours of the next day in GMT. Second, you have to add SFhour to 24 to
    // give a valid hour in the 24-hour clock.
    if (SFhour < 0){
        day = day - 1;
        // If setting the day count back one would make it negative, it's Saturday,
        // so set the count to 6.
        if (day < 0){
            day = 6;
        }
        SFhour = SFhour + 24;
    }

    // Now determine which photo to show based on whether it's a workday or a
    // weekend; what time it is; and, if it's a time and day when Kent is
    // working, what platform the user is on.

    //If it's not Sunday
    if (day != 0){
        //And it's between 10am and noon, inclusive
        if (SFhour >= 10 && SFhour <= 12){
            imageRef = "images/kent_tiredAndIrritated.jpg";
        } //Or else it's between 1pm and 3pm, inclusive
        }else if (SFhour >= 13 && SFhour <= 15){
            imageRef = "images/kent_hungry.jpg";
        } //Or else it's between 4pm and 5pm, inclusive
        }else if (SFhour >= 16 && SFhour <= 17){
            //If user is on Mac, show Kent working on Mac
            if (platform == "MacPPC"){

```



```

        imageRef = "images/kent_gettingStartedOnMac.jpg";
        //If user is on Win, show Kent working on Win
    }else{
        imageRef = "images/kent_gettingStartedOnWin.jpg";
    }
    //Or else it's after 6pm but before the stroke of midnight
    }else if (SFhour >= 18){
        //If it's Saturday
        if (day == 6){
            imageRef = "images/kent_dancing.jpg";
        }
        //If it's not Saturday, check the user's platform
    }else if (platform == "MacPPC"){
        imageRef = "images/kent_hardAtWorkOnMac.jpg";
    }else{
        imageRef = "images/kent_hardAtWorkOnWin.jpg";
    }
    }else{
        imageRef = "images/kent_sleeping.jpg";
    }
    }
    //If it's after midnight and before 10am, or anytime on Sunday
    }else{
        imageRef = "images/kent_sleeping.jpg";
    }
    }

    return imageRef;
}

</script>
</head>

<body>
</body>
</html>

```

## Creating property inspectors for locked content

Once you've created a translator, you'll need to create a property inspector for the content so that the user can change its properties (for example, the file to be included, or one of the conditions in a conditional statement). Inspecting translated content is a unique problem for several reasons:

- The user may want to change the properties of the translated content, and those changes must be reflected in the untranslated content.
- The DOM contains the translated content (that is, the lock tags and the tags they surround are nodes in the DOM), but the `outerHTML` property of the `documentElement` and the `dreamweaver.getSelection()` and `dreamweaver.nodeToOffsets()` functions act on the untranslated source.
- The tags you're inspecting are different before and after translation.

If the user might disable your translator, and another inspector is not already handling the untranslated tags, you may need to create two property inspectors—one for the untranslated tag or tags, and one for the locked region created by translation. The interfaces of the two inspectors can be identical, but the comment at the top of the file and the `canInspectSelection()` and `inspectSelection()` functions—must differ.

For example, an inspector for the HAPPY tag might have a comment that looks like this:

```
<!-- tag:HAPPY,priority:5,selection:exact,hline,vline -->
```

The inspector for the translated HAPPY tag, however, would have a comment that looks like this:

```
<!-- tag:*LOCKED*,priority:5,selection:within,hline,vline -->
```

The `canInspectSelection()` function for the untranslated HAPPY inspector is simple: since the selection type is exact, it can return `TRUE` without further analysis. For the translated HAPPY inspector, this function is more complicated; the keyword `*LOCKED*` indicates that the inspector is appropriate when the selection is within a locked region, but because a document could have several locked regions, further checks must be performed to determine if the inspector matches this particular locked region.

Yet another problem is inherent in inspecting translated content. When you call `dom.getSelection()`, the values returned by default are offsets into the untranslated source. To expand the selection properly so that the locked region (and only the locked region) is selected, use the following method:

```
var currentDOM = dw.getDocumentDOM();
var offsets = currentDOM.getSelection();
var theSelection = currentDOM.offsetsToNode(offsets[0],offsets[0]+1);
```

Using `offsets[0]+1` as the second argument assures that you remain within the opening lock tag when you convert the offsets to a node. If you use `offsets[1]` as the second argument, you risk selecting the node above the lock.

Once you have the selection (after first making sure that its `nodeType` is `node.ELEMENT_NODE`), you can inspect the type attribute to see if the locked region matches this inspector. For example:

```
if (theSelection.nodeType == node.ELEMENT_NODE && theSelection.getAttribute('type')
== 'happy'){
    return true;
}else{
    return false
}
```

To populate the fields in the inspector for the translated tag, you must parse the value of the orig attribute. For example, if the untranslated code is <HAPPY TIME="22"> and the property inspector has a field labeled Time, you must extract the value of the TIME attribute from the orig string.

```
function inspectSelection() {
    var currentDOM = dw.getDocumentDOM();
    var currSelection = currentDOM.getSelection();
    var theObj = currentDOM.offsetsToNode(currSelection[0],currSelection[0]+1);

    if (theObj.nodeType != Node.ELEMENT_NODE) {
        return;
    }

    // To convert the encoded characters back to their
    // original values, use the unescape() method.
    var origAtt = unescape(theObj.getAttribute("ORIG"));

    // Convert the string to lower case for processing
    var origAttLC = origAtt.toLowerCase();

    var timeStart = origAttLC.indexOf("time=");
    var timeEnd = origAttLC.indexOf('"',timeStart+6);
    var timeValue = origAtt.substr(timeStart+6,timeEnd);

    document.layers['timelayer'].document.timeForm.timefield.value = timeValue;
}
```

Once you've parsed the orig attribute in order to populate the fields in the property inspector for the translated tag, your first instinct is probably to set the value of the orig attribute if the user changes the value in any of the fields. This is not the best course of action, however, as you are likely to run into the restriction against making changes in a locked region. You can avoid this problem by changing the original markup and then retranslating.

The property inspector for translated server-side includes (Configuration/Inspectors/ssi\_translated.js) demonstrates this technique in its setComment() function. Rather than rewriting the orig attribute, the inspector assembles a new SSI comment. It then inserts that comment into the document in place of the old one by rewriting the entire document contents, which in turn generates a new orig attribute. The following code summarizes this technique:

```
// Assemble the new include comment. radioStr and URL are
// variables defined earlier in the code.
newInc = "<!--#include " + radioStr + "=" + "" + URL + "" + " -->";

// Get the contents of the document.
var entireDocObj = dreamweaver.getDocumentDOM();
var docSrc = entireDocObj.documentElement.outerHTML;

// Store everything up to the SSI comment and everything after
// the SSI comment in the beforeSelStr and afterSelStr variables.
var beforeSelStr = docSrc.substring(0, curSelection[0] );
var afterSelStr = docSrc.substring(curSelection[1]);

// Assemble the new contents of the document.
docSrc = beforeSelStr + newInc + afterSelStr;

// Set the outerHTML of the HTML tag (represented by
// the documentElement object) to the new contents,
// and then set the selection back to the locked region
// surrounding the SSI comment.
entireDocObj.documentElement.outerHTML = docSrc;
entireDocObj.setSelection(curSelection[0], curSelection[0]+1);
```

## Finding bugs in your translator

If the `translateMarkup()` function contains certain types of errors, the translator loads properly, but it fails silently when invoked. Although failing silently prevents Dreamweaver from becoming unstable, it can hinder development, especially when trying to find one small syntax error in 10 or 20 or 30 lines of code.

If your translator fails, one debugging method that works well is to turn the translator into a command, as follows:

- 1 Copy the entire contents of the translator file to a new document, and save it in the Configuration/Commands folder inside the Dreamweaver application folder.

- 2 At the top of the document, between the `SCRIPT` tags, add the following function:

```
function commandButtons(){  
    return new Array( "OK","translateMarkup(dreamweaver.getDocumentPath('document'),  
dreamweaver.getSiteRoot(),  
dreamweaver.getDocumentDOM().documentElement.outerHTML);  
window.close()", "Cancel", "window.close()");  
}
```

- 3 At the end of the `translateMarkup()` function, comment out the `return whateverTheReturnValues` line and replace it with `dreamweaver.getDocumentDOM().documentElement.outerHTML = whateverTheReturnValues`. For example:

```
// return theCode;  
dreamweaver.getDocumentDOM().documentElement.outerHTML = theCode;  
}  
/* end of translateMarkup() */
```

- 4 In the BODY of the document, add a form with no input fields. For example:

```
<body>  
<form>  
Hello.  
</form>  
</body>
```

- 5 Restart Dreamweaver, and then choose your “translator command” from the Commands menu. When you click OK, the `translateMarkup()` function is called, simulating translation.

If you see no error message and translation still fails, you probably have a logic error in your code.

- 6 Add `alert()` statements in strategic spots throughout the `translateMarkup()` function so you can make sure you're hitting the proper branches, and so you can check the values of variables and properties at different points. For example:

```
for (var i=0; i< foo.length; i++){  
  alert("we're at the top of the foo.length array, and the value of i is " + i);  
  /* rest of loop */  
}
```

- 7 After adding in the `alert()` statements, choose your command from the Commands menu, click Cancel, and then choose it again. This reloads the command file, incorporating your changes.

# INDEX

## A

- addBehavior() 22
  - dreamweaver.timelineInspector.addBehavior() 130
- addFrame() 130
- addKeyframe() 130
- addLinkToExistingFile() 109
- addLinkToNewFile() 109
- addObject() 131
- addTimeline() 131
- alert() 10
- align() 91
- applyBehavior() 264
- applyCharacterMarkup() 61
- applyCSSStyle() 39
- applyFontMarkup() 61
- applyHTMLStyle() 82
- applyTemplate() 93
- appName property 13
- appVersion property 13
- arguments
  - optional 20
  - passed from menuitem 236
  - receiveArguments() 238
- arrange() 91
- arrangeFloatingPalettes() 149
- array object 10
- arrowDown() 85
- arrowLeft() 85
- arrowRight() 86
- arrowUp() 86
- attribute translators 277
  - creating 278
  - debugging 293
  - sample code 279
- attributes property 15

## B

- backspaceKey() 86
- behaviorFunction() 265
- behaviors
  - API 263
  - helper functions 263
  - inserting multiple functions with 263
  - required functions 263
  - sample code 270
  - user experience 262
- block/tag translators 277
  - debugging 293
  - sample code 285
- blur() 10
- body property 14
- books, JavaScript 8
- boolean object 10
- bringDWToFront() 193
- bringFWToFront() 193
- browseDocument() 44
- browseForFileURL() 48
- browseForFolderURL() 48
- button object 10

## C

- C functions
  - calling from JavaScript 218
  - in mm\_jsapi.h 211
- canAcceptBehavior() 266
- canAcceptCommand()
  - in menu commands 237
  - in regular commands 229
- canAddFrame() 169
- canAddKeyFrame() 169
- canAddLinkToFile() 171
- canAlign() 158
- canArrange() 159
- canChangeLink() 171
- canChangeObject() 170
- canCheckIn() 171

- canCheckOut() 172
- canClipCopy() 165
- canClipCopyText() 159
- canClipCut() 165
- canClipPaste()
  - dom.canClipPaste() 159
  - dreamweaver.canClipPaste 165
- canClipPasteText() 159
- canConnect() 172
- canConvertLayersToTable() 159
- canConvertTablesToLayers() 160
- canDecreaseColspan() 160
- canDecreaseRowspan() 160
- canDeleteTableColumn() 160
- canDeleteTableRow() 160
- canEditNoFramesContent() 161
- canEditSelection() 169
- canExportCSS() 166
- canFindLinkSource() 172
- canFindNext() 166
- canGet() 173
- canIncreaseColspan() 161
- canIncreaseRowspan() 161
- canInsertTableColumns() 161
- canInsertTableRows() 161
- canInspectSelection() 247
- canLocateInSite() 173
- canMakeEditable() 173
- canMakeNewEditableRegion() 162
- canMakeNewFileOrFolder() 174
- canMarkSelectionAsEditable() 162
- canMergeTableCells() 162
- canOpen() 174
- canOpenInFrame() 166
- canPlayPlugin() 162
- canPlayRecordedCommand() 166
- canPut() 174
- canRecreateCache() 174
- canRedo()
  - dom.canRedo() 162
  - dreamweaver.canRedo() 166
- canRefresh() 175
- canRemoveEditableRegion() 163
- canRemoveFrame() 170
- canRemoveKeyFrame() 170
- canRemoveLink() 175
- canRemoveObject() 170
- canRevertDocument() 167
- canSaveAll() 167
- canSaveDocument() 167
- canSaveDocumentAsTemplate() 167
- canSaveFrameset() 168
- canSaveFramesetAs() 168
- canSelectAll() 168
- canSelectNewer() 175
- canSelectTable() 163
- canSetLayout() 175
- canSetLinkHref() 163
- canShowFindDialog() 168
- canShowListPropertiesDialog() 163
- canSplitFrame() 164
- canSplitTableCell() 164
- canStopPlugin() 164
- canSynchronize() 176
- canUndo()
  - dom.canUndo() 164
  - dreamweaver.canUndo() 169
- canUndoCheckOut() 176
- canViewAsRoot() 176
- changeLink() 110
- changeLinkSitewide() 109
- changeObject() 131
- checkbox object 10
- checkIn() 110
- checkLinks() 110
- checkOut() 111
- checkSpelling() 74
- checkTargetBrowsers()
  - dom.checkTargetBrowsers() 74
  - site.checkTargetBrowsers() 111
- childNodes property
  - of comment objects 17
  - of document objects 14
  - of tag objects 15
  - of text objects 17
- clearInterval() 10
- clearSteps() 78
- clearTemp() 203
- clearTimeout() 10
- clipCopy()
  - dom.clipCopy() 33
  - dreamweaver.clipCopy() 35
- clipCopyText() 33



- clipCut()
  - dom.clipCut() 33
  - dreamweaver.clipCut() 36
- clipPaste()
  - dom.clipPasteText() 34
  - dreamweaver.clipPaste() 36
- clipPasteText() 35
- close()
  - MMNotes.close() 185
  - window.close() 10
- closeDocument() 49
- CloseNotesFile() 188
- commandButtons()
  - in menu commands 237
  - in regular commands 230
- commands
  - adding to menus 233
  - API 229
  - sample code 232
  - user experience 228
- comment object 17
- confirm() 10
- convertLayersToTable() 38
- convertTablesToLayers() 38
- convertTo30() 38
- convertWidthsToPercent() 125
- convertWidthsToPixels() 125
- copy() 178
- copySteps() 78
- createDocument() 49
- createFolder() 178

## D

- data property
  - of comment objects 17
  - of httpReply objects 202
  - of text objects 17
- data translators
  - API 275
  - debugging 293
  - for attributes 278
  - for tags or blocks of code 284
  - kinds of 277
  - required functions 275
  - user experience 274
- date object 10
- decreaseColspan() 125

- decreaseRowspan() 125
- defineSites() 111
- deleteBehavior() 267
- deleteKey() 87
- deleteSelectedItem() 97
- deleteSelectedStyle()
  - dreamweaver.cssStylePalette.deleteSelectedStyle() 40
  - dreamweaver.htmlStylePalette.deleteSelectedStyle() 82
  - dw.htmlStylePalette.deleteSelectedStyle() 82
- deleteSelectedTemplate() 98
- deleteSelection()
  - dom.deleteSelection() 61
  - dreamweaver.deleteSelection() 71
  - site.deleteSelection() 112
- deleteTableColumn() 126
- deleteTableRow() 126
- Design Notes
  - C API 188
  - file structure 184
  - JavaScript API 184
  - user experience 184
- detachFromLibrary() 93
- detachFromTemplate() 94
- displayHelp()
  - in behavior actions 266
  - in object files 223
  - in property inspector files 248
- document node 14
- document object
  - DOM Level 1 properties and methods of 14
  - Netscape DOM properties and methods of 10
- document object model
  - DOM Level 1 specification 10
  - in Dreamweaver 10
- documentEdited() 255
- documentElement property 14
- doDeferredTableUpdate() 126
- DOM
  - DOM Level 1 specification 10
  - dom object 20
  - in Dreamweaver 10
- dom.addBehavior() 22
- dom.align() 91
- dom.applyCharacterMarkup() 61
- dom.applyCSSStyle() 39
- dom.applyFontMarkup() 61
- dom.applyHTMLStyle() 82

- dom.applyTemplate() 93
- dom.arrange() 91
- dom.arrowDown() 85
- dom.arrowLeft() 85
- dom.arrowRight() 86
- dom.arrowUp() 86
- dom.backspaceKey() 86
- dom.canAlign() 158
- dom.canArrange() 159
- dom.canClipCopyText() 159
- dom.canClipPaste() 159
- dom.canClipPasteText() 159
- dom.canConvertLayersToTable() 159
- dom.canConvertTablesToLayers() 160
- dom.canDecreaseColspan() 160
- dom.canDecreaseRowspan() 160
- dom.canDeleteTableColumn() 160
- dom.canDeleteTableRow() 160
- dom.canEditNoFramesContent() 161
- dom.canIncreaseColspan() 161
- dom.canIncreaseRowspan() 161
- dom.canInsertTableColumns() 161
- dom.canInsertTableRows() 161
- dom.canMakeNewEditableRegion() 162
- dom.canMarkSelectionAsEditable() 162
- dom.canMergeTableCells() 162
- dom.canPlayPlugin() 162
- dom.canRedo() 162
- dom.canRemoveEditableRegion() 163
- dom.canSelectTable() 163
- dom.canSetLinkHref() 163
- dom.canShowListPropertiesDialog() 163
- dom.canSplitFrame() 164
- dom.canSplitTableCell() 164
- dom.canStopPlugin() 164
- dom.canUndo() 164
- dom.checkSpelling() 74
- dom.checkTargetBrowsers() 74
- dom.clipCopy 33
- dom.clipCopyText() 33
- dom.clipCut() 33
- dom.clipPaste 34
- dom.clipPasteText() 35
- dom.convertLayersToTable() 38
- dom.convertTablesToLayers() 38
- dom.convertTo30() 38
- dom.convertWidthsToPercent() 125
- dom.convertWidthsToPixels() 125
- dom.decreaseColspan() 125
- dom.decreaseRowspan() 125
- dom.deleteKey() 87
- dom.deleteSelection() 61
- dom.deleteTableColumn() 126
- dom.deleteTableRow() 126
- dom.detachFromLibrary() 93
- dom.detachFromTemplate() 94
- dom.doDeferredTableUpdate() 126
- dom.editAttribute() 62
- dom.endOfDocument() 87
- dom.endOfLine() 87
- dom.exitBlock() 62
- dom.getAttachedTemplate() 94
- dom.getBehavior() 23
- dom.getEditableRegionList() 94
- dom.getEditableRetionList() 95
- dom.getEditNoFramesContent() 135
- dom.getFocus() 150
- dom.getFontMarkup() 62
- dom.getFrameNames() 60
- dom.getIsLibraryDocument() 94
- dom.getIsTemplateDocument() 95
- dom.getLinkHref() 62
- dom.getLinkTarget() 63
- dom.getListTag() 63
- dom.getPreventLayerOverlaps() 135
- dom.getRulerOrigin() 146
- dom.getRulerUnits() 146
- dom.getSelectedEditableRegion() 95
- dom.getSelectedNode() 105
- dom.getSelection() 105
- dom.getShowFrameBorders() 135
- dom.getShowGrid() 135
- dom.getShowHeaderView() 136
- dom.getShowImageMaps() 136
- dom.getShowLayerBorders() 136
- dom.getShowRulers() 136
- dom.getShowTableBorders() 137
- dom.getShowTracingImage() 137
- dom.getSnapToGrid() 137
- dom.getTableExtent() 126
- dom.getTextAlignment() 63
- dom.getTextFormat() 63
- dom.getTracingImageOpacity() 146
- dom.getWindowTitle() 150

- dom.hasCharacterMarkup() 64
- dom.hasTracingImage() 164
- dom.increaseColspan() 127
- dom.increaseRowspan() 127
- dom.indent() 64
- dom.insertHTML() 65
- dom.insertLibraryItem() 95
- dom.insertObject() 65
- dom.insertTableColumns() 127
- dom.insertTableRows() 127
- dom.insertText() 66
- dom.loadTracingImage() 147
- dom.makeSizesEqual() 91
- dom.markSelectionAsEditable() 96
- dom.mergeTableCells() 128
- dom.moveSelectionBy() 92
- dom.newBlock() 66
- dom.newEditableRegion() 96
- dom.nextParagraph() 88
- dom.nextWord() 88
- dom.nodeToOffsets() 106
- dom.offsetsToNode() 106
- dom.outdent() 67
- dom.pageDown() 88
- dom.pageUp() 89
- dom.playAllPlugins() 147
- dom.playPlugin() 147
- dom.previousParagraph() 89
- dom.previousWord() 89
- dom.reapplyBehaviors() 23
- dom.redo() 76
- dom.removeAllTableHeights() 128
- dom.removeAllTableWidths() 128
- dom.removeBehavior() 24
- dom.removeCharacterMarkup() 67
- dom.removeCSSStyle() 40
- dom.removeEditableRegion() 96
- dom.removeFontMarkup() 67
- dom.removeLink() 67
- dom.resizeSelection() 68
- dom.resizeSelectionBy() 92
- dom.runTranslator() 144
- dom.selectAll() 107
- dom.selectChild() 103
- dom.selectParent() 103
- dom.selectTable() 107
- dom.setAttributeWithErrorChecking() 68
- dom.setEditNoFramesContent() 137
- dom.setLayerTag() 93
- dom.setLinkHref() 68
- dom.setLinkTarget() 69
- dom.setListBoxKind() 69
- dom.setListTag() 70
- dom.setPreventLayerOverlaps() 138
- dom.setRulerOrigin() 147
- dom.setRulerUnits() 148
- dom.setSelectedNode() 107
- dom.setSelection() 108
- dom.setShowFrameBorders() 138
- dom.setShowGrid() 138
- dom.setShowHeaderView() 138
- dom.setShowImageMaps() 139
- dom.setShowLayerBorders() 139
- dom.setShowRulers() 139
- dom.setShowTableBorders() 139
- dom.setShowTracingImage() 140
- dom.setSnapToGrid() 140
- dom.setTableCellTag() 128
- dom.setTableColumns() 129
- dom.setTableRows() 129
- dom.setTextAlignment() 70
- dom.setTextFieldKind() 70
- dom.setTracingImageOpacity() 148
- dom.setTracingImagePosition() 148
- dom.showFontColorDialog() 70
- dom.showInsertTableRowsOrColumnsDialog() 129
- dom.showListPropertiesDialog() 69
- dom.showPagePropertiesDialog() 75
- dom.snapTracingImageToSelection() 148
- dom.splitFrame() 60
- dom.splitTableCell() 129
- dom.startOfDocument() 90
- dom.startOfLine() 90
- dom.stopAllPlugins() 149
- dom.stopPlugin() 149
- dom.stripTag() 103
- dom.undo() 76
- dom.updateCurrentPage() 96
- dom.wrapTag() 104
- dreamweaver object
  - methods of 20
  - properties of 13
- dreamweaver.arrangeFloatingPalettes() 149
- dreamweaver.behaviorInspector object 22

dreamweaver.behaviorInspector.getBehaviorAt() 28  
 dreamweaver.behaviorInspector.getBehaviorCount() 28  
 dreamweaver.behaviorInspector.getSelectedBehavior() 29  
 dreamweaver.behaviorInspector.moveBehaviorDown() 30  
 dreamweaver.behaviorInspector.moveBehaviorUp() 31  
 dreamweaver.behaviorInspector.setSelectedBehavior() 32  
 dreamweaver.browseDocument() 44  
 dreamweaver.browseForFileURL() 48  
 dreamweaver.browseForFolderURL() 48  
 dreamweaver.canClipCopy() 165  
 dreamweaver.canClipCut() 165  
 dreamweaver.canClipPaste() 165  
 dreamweaver.canExportCSS() 166  
 dreamweaver.canFindNext() 166  
 dreamweaver.canOpenInFrame() 166  
 dreamweaver.canPlayRecordedCommand() 166  
 dreamweaver.canRedo() 166  
 dreamweaver.canRevertDocument() 167  
 dreamweaver.canSaveAll() 167  
 dreamweaver.canSaveDocument() 167  
 dreamweaver.canSaveDocumentAsTemplate() 167  
 dreamweaver.canSaveFrameset() 168  
 dreamweaver.canSaveFramesetAs() 168  
 dreamweaver.canSelectAll() 168  
 dreamweaver.canShowFindDialog() 168  
 dreamweaver.canUndo() 169  
 dreamweaver.clipCopy() 35  
 dreamweaver.clipCut() 36  
 dreamweaver.clipPaste() 36  
 dreamweaver.closeDocument() 49  
 dreamweaver.createDocument() 49  
 dreamweaver.cssStylePalette object 39  
 dreamweaver.cssStylePalette.deleteSelectedStyle() 40  
 dreamweaver.cssStylePalette.duplicateSelectedStyle() 40  
 dreamweaver.cssStylePalette.editSelectedStyle() 41  
 dreamweaver.cssStylePalette.editStyleSheet() 41  
 dreamweaver.cssStylePalette.getSelectedStyle() 41  
 dreamweaver.cssStylePalette.getSelectedTarget() 42  
 dreamweaver.cssStylePalette.getStyles() 43  
 dreamweaver.cssStylePalette.newStyle() 43  
 dreamweaver.deleteSelection() 71  
 dreamweaver.editCommandList() 37  
 dreamweaver.editFontList() 71  
 dreamweaver.editLockedRegions() 144  
 dreamweaver.exportCSS() 49  
 dreamweaver.exportEditableRegionsAsXML() 50  
 dreamweaver.findNext() 55  
 dreamweaver.getActiveWindow() 150  
 dreamweaver.getBehaviorElement() 25  
 dreamweaver.getBehaviorEvent() 153  
 dreamweaver.getBehaviorTag() 26  
 dreamweaver.getBrowserList() 45  
 dreamweaver.getClipboardText() 36  
 dreamweaver.getConfigurationPath() 101  
 dreamweaver.getDocumentDOM() 21  
 dreamweaver.getDocumentList() 151  
 dreamweaver.getDocumentPath() 101  
 dreamweaver.getElementRef() 75  
 dreamweaver.getExtensionEditorList() 45  
 dreamweaver.getFloaterVisibility() 151  
 dreamweaver.getFocus() 151  
 dreamweaver.getFontList() 71  
 dreamweaver.getHideAllFloaters() 140  
 dreamweaver.getKeyState() 72  
 dreamweaver.getMenuNeedsUpdating() 99  
 dreamweaver.getObjectRefs() 154  
 dreamweaver.getObjectTags() 155  
 dreamweaver.getPrimaryBrowser() 45  
 dreamweaver.getPrimaryExtensionEditor() 46  
 dreamweaver.getRecentFileList() 50  
 dreamweaver.getRedoText() 76  
 dreamweaver.getSecondaryBrowser() 46  
 dreamweaver.getShowDialogsOnInsert() 73  
 dreamweaver.getShowInvisibleElements() 140  
 dreamweaver.getShowStatusBar() 141  
 dreamweaver.getSiteRoot() 102  
 dreamweaver.getTokens() 123  
 dreamweaver.getTranslatorList() 145  
 dreamweaver.getUndoText() 77  
 dreamweaver.historyPalette object 76  
 dreamweaver.historyPalette.clearSteps() 78  
 dreamweaver.historyPalette.copySteps() 78  
 dreamweaver.historyPalette.getSelectedSteps() 79  
 dreamweaver.historyPalette.getStepCount() 79  
 dreamweaver.historyPalette.getStepsAsJavaScript() 80  
 dreamweaver.historyPalette.getUndoState() 80  
 dreamweaver.historyPalette.replaySteps() 81  
 dreamweaver.historyPalette.saveAsCommand() 81  
 dreamweaver.historyPalette.setSelectedSteps() 81  
 dreamweaver.historyPalette.setUndoState() 82  
 dreamweaver.htmlStylePalette object 82  
 dreamweaver.htmlStylePalette.canEditSelection() 169  
 dreamweaver.htmlStylePalette.deleteSelectedStyle() 82  
 dreamweaver.htmlStylePalette.duplicateSelectedStyle() 83

dreamweaver.htmlStylePalette.editSelectedStyle() 83  
 dreamweaver.htmlStylePalette.getSelectedStyle() 83  
 dreamweaver.htmlStylePalette.getStyles() 83  
 dreamweaver.htmlStylePalette.newStyle() 84  
 dreamweaver.htmlStylePalette.setSelectedStyle() 84  
 dreamweaver.importXMLIntoTemplate() 50  
 dreamweaver.isRecording() 169  
 dreamweaver.latin1ToNative() 124  
 dreamweaver.libraryPalette object 93  
 dreamweaver.libraryPalette.deleteSelectedItem() 97  
 dreamweaver.libraryPalette.getSelectedItem() 97  
 dreamweaver.libraryPalette.newFromDocument() 97  
 dreamweaver.libraryPalette.recreateFromDocument() 98  
 dreamweaver.libraryPalette.renameSelectedItem() 98  
 dreamweaver.nativeToLatin1() 124  
 dreamweaver.newFromTemplate() 50  
 dreamweaver.nodeToOffsets() 156  
 dreamweaver.notifyMenuUpdated() 100  
 dreamweaver.offsetsToNode() 157  
 dreamweaver.openDocument() 51  
 dreamweaver.openDocumentFromSite() 51  
 dreamweaver.openInFrame() 52  
 dreamweaver.openWithApp() 46  
 dreamweaver.openWithBrowseDialog() 47  
 dreamweaver.openWithExternalTextEditor() 47  
 dreamweaver.openWithImageEditor() 47  
 dreamweaver.playRecordedCommand() 77  
 dreamweaver.popupAction() 27  
 dreamweaver.popupCommand() 157  
 dreamweaver.quitApplication() 73  
 dreamweaver.redo() 77  
 dreamweaver.relativeToAbsoluteURL() 102  
 dreamweaver.releaseDocument() 52  
 dreamweaver.reloadMenus() 100  
 dreamweaver.replace() 55  
 dreamweaver.replaceAll() 55  
 dreamweaver.revertDocument() 52  
 dreamweaver.runCommand() 37  
 dreamweaver.saveAll() 53  
 dreamweaver.saveDocument() 53  
 dreamweaver.saveDocumentAs() 53  
 dreamweaver.saveDocumentAsTemplate() 54  
 dreamweaver.saveFrameset() 54  
 dreamweaver.saveFramesetAs() 54  
 dreamweaver.selectAll() 108  
 dreamweaver.setActiveWindow() 152  
 dreamweaver.setFloaterVisibility() 152  
 dreamweaver.setHideAllFloaters() 141  
 dreamweaver.setSelection() 158  
 dreamweaver.setShowInvisibleElements() 141  
 dreamweaver.setShowStatusBar() 141  
 dreamweaver.setUpComplexFind() 56  
 dreamweaver.setUpComplexFindReplace() 57  
 dreamweaver.setUpFind() 58  
 dreamweaver.setUpFindReplace() 59  
 dreamweaver.showAboutBox() 73  
 dreamweaver.showFindDialog() 59  
 dreamweaver.showFindReplaceDialog() 60  
 dreamweaver.showGridSettingsDialog() 149  
 dreamweaver.showPreferencesDialog() 74  
 dreamweaver.showProperties() 152  
 dreamweaver.showQuickTagEditor() 104  
 dreamweaver.startRecording() 77  
 dreamweaver.stopRecording() 78  
 dreamweaver.templatePalette object 93  
 dreamweaver.templatePalette.deleteSelectedTemplate() 98  
 dreamweaver.templatePalette.getSelectedTemplate() 98  
 dreamweaver.templatePalette.newBlankTemplate() 99  
 dreamweaver.templatePalette.renameSelectedTemplate() 99  
 dreamweaver.timelineInspector object 130  
 dreamweaver.timelineInspector.addBehavior() 130  
 dreamweaver.timelineInspector.addFrame() 130  
 dreamweaver.timelineInspector.addKeyframe() 130  
 dreamweaver.timelineInspector.addObject() 131  
 dreamweaver.timelineInspector.addTimeline() 131  
 dreamweaver.timelineInspector.canAddFrame() 169  
 dreamweaver.timelineInspector.canAddKeyFrame() 169  
 dreamweaver.timelineInspector.canChangeObject() 170  
 dreamweaver.timelineInspector.canRemoveFrame() 170  
 dreamweaver.timelineInspector.canRemoveKeyFrame() 170  
 dreamweaver.timelineInspector.canRemoveObject() 170  
 dreamweaver.timelineInspector.changeObject() 131  
 dreamweaver.timelineInspector.getAutoplay() 131  
 dreamweaver.timelineInspector.getCurrentFrame() 132  
 dreamweaver.timelineInspector.getLoop() 132  
 dreamweaver.timelineInspector.recordPathOfLayer() 132  
 dreamweaver.timelineInspector.removeBehavior() 132  
 dreamweaver.timelineInspector.removeFrame() 133  
 dreamweaver.timelineInspector.removeKeyframe() 133  
 dreamweaver.timelineInspector.removeObject() 133  
 dreamweaver.timelineInspector.removeTimeline() 133  
 dreamweaver.timelineInspector.renameTimeline() 134  
 dreamweaver.timelineInspector.setAutoplay() 134  
 dreamweaver.timelineInspector.setCurrentFrame() 134

dreamweaver.timelineInspector.setLoop() 134  
 dreamweaver.toggleFloater() 153  
 dreamweaver.undo() 78  
 dreamweaver.updatePages() 97  
 dreamweaver.useTranslatedSource() 145  
 duplicateSelectedStyle()  
     dreamweaver.cssStylePalette.duplicateSelectedStyle() 40  
     dreamweaver.htmlStylePalette.duplicateSelectedStyle() 83  
 dw object 20  
 dw.arrangeFloatingPalettes() 149  
 dw.behaviorInspector.getBehaviorAt() 28  
 dw.behaviorInspector.getBehaviorCount() 28  
 dw.behaviorInspector.getSelectedBehavior() 29  
 dw.behaviorInspector.moveBehaviorDown() 30  
 dw.behaviorInspector.moveBehaviorUp() 31  
 dw.behaviorInspector.setSelectedBehavior() 32  
 dw.browseDocument() 44  
 dw.browseForFileURL() 48  
 dw.browseForFolderURL() 48  
 dw.canClipCopy() 165  
 dw.canClipCut() 165  
 dw.canClipPaste() 165  
 dw.canExportCSS() 166  
 dw.canFindNext() 166  
 dw.canOpenInFrame() 166  
 dw.canPlayRecordedCommand() 166  
 dw.canRedo() 166  
 dw.canRevertDocument() 167  
 dw.canSaveAll() 167  
 dw.canSaveDocument() 167  
 dw.canSaveDocumentAsTemplate() 167  
 dw.canSaveFrameset() 168  
 dw.canSaveFramesetAs() 168  
 dw.canSelectAll() 168  
 dw.canShowFindDialog() 168  
 dw.canUndo() 169  
 dw.clipCopy() 35  
 dw.clipCut() 36  
 dw.clipPaste() 36  
 dw.closeDocument() 49  
 dw.createDocument() 49  
 dw.cssStylePalette.deleteSelectedStyle() 40  
 dw.cssStylePalette.duplicateSelectedStyle() 40  
 dw.cssStylePalette.editSelectedStyle() 41  
 dw.cssStylePalette.editStyleSheet() 41  
 dw.cssStylePalette.getSelectedStyle() 41  
 dw.cssStylePalette.getSelectedTarget() 42  
 dw.cssStylePalette.getStyles() 43  
 dw.cssStylePalette.newStyle() 43  
 dw.deleteSelection() 71  
 dw.editCommandList() 37  
 dw.editFontList() 71  
 dw.editLockedRegions() 144  
 dw.exportCSS() 49  
 dw.exportEditableRegionsAsXML() 50  
 DWfile DLL  
     API 178  
         checking for 177  
 DWfile.copy() 178  
 DWfile.createFolder() 178  
 DWfile.exists() 179  
 DWfile.getAttributes() 179  
 DWfile.getModificationDate() 180  
 DWfile.listFolder() 181  
 DWfile.read() 181  
 DWfile.remove() 182  
 DWfile.write() 182  
 dw.findNext() 55  
 dw.getActiveWindow() 150  
 dw.getBehaviorElement() 25  
 dw.getBehaviorTag() 26  
 dw.getBrowserList() 45  
 dw.getClipboardText() 36  
 dw.getConfigurationPath() 101  
 dw.getDocumentDOM() 21  
 dw.getDocumentList() 151  
 dw.getDocumentPath() 101  
 dw.getElementRef() 75  
 dw.getExtensionEditorList() 45  
 dw.getFloaterVisibility() 151  
 dw.getFocus() 151  
 dw.getFontList() 71  
 dw.getHideAllFloaters() 140  
 dw.getKeyState() 72  
 dw.getMenuNeedsUpdating() 99  
 dw.getPrimaryBrowser() 45  
 dw.getPrimaryExtensionEditor() 46  
 dw.getRecentFileList() 50  
 dw.getRedoText() 76  
 dw.getSecondaryBrowser() 46  
 dw.getShowDialogsOnInsert() 73  
 dw.getShowInvisibleElements() 140  
 dw.getShowStatusBar() 141  
 dw.getSiteRoot() 102

dw.getTokens() 123  
 dw.getTranslatorList() 145  
 dw.getUndoText() 77  
 dw.historyPalette.clearSteps() 78  
 dw.historyPalette.copySteps() 78  
 dw.historyPalette.getSelectedSteps() 79  
 dw.historyPalette.getStepCount() 79  
 dw.historyPalette.getStepsAsJavaScript() 80  
 dw.historyPalette.getUndoState() 80  
 dw.historyPalette.replaySteps() 81  
 dw.historyPalette.saveAsCommand() 81  
 dw.historyPalette.setSelectedSteps() 81  
 dw.historyPalette.setUndoState() 82  
 dw.htmlStylePalette.canEditSelection() 169  
 dw.htmlStylePalette.deleteSelectedStyle() 82  
 dw.htmlStylePalette.duplicateSelectedStyle() 83  
 dw.htmlStylePalette.editSelectedStyle() 83  
 dw.htmlStylePalette.getSelectedStyle() 83  
 dw.htmlStylePalette.getStyles() 83  
 dw.htmlStylePalette.newStyle() 84  
 dw.htmlStylePalette.setSelectedStyle() 84  
 dw.importXMLIntoTemplate() 50  
 dw.isRecording() 169  
 dw.latin1ToNative() 124  
 dw.libraryPalette.deleteSelectedItem() 97  
 dw.libraryPalette.getSelectedItem() 97  
 dw.libraryPalette.newFromDocument() 97  
 dw.libraryPalette.recreateFromDocument() 98  
 dw.libraryPalette.renameSelectedItem() 98  
 dw.nativeToLatin1() 124  
 dw.newFromTemplate() 50  
 dw.notifyMenuUpdated() 100  
 dw.openDocument() 51  
 dw.openDocumentFromSite() 51  
 dw.openInFrame() 52  
 dw.openWithApp() 46  
 dw.openWithBrowseDialog() 47  
 dw.openWithExternalTextEditor() 47  
 dw.openWithImageEditor() 47  
 dw.playRecordedCommand() 77  
 dw.popupAction() 27  
 dw.quitApplication() 73  
 dw.redo() 77  
 dw.relativeToAbsoluteURL() 102  
 dw.releaseDocument() 52  
 dw.reloadMenus() 100  
 dw.replace() 55  
 dw.replaceAll() 55  
 dw.revertDocument() 52  
 dw.runCommand() 37  
 dw.saveAll() 53  
 dw.saveDocument() 53  
 dw.saveDocumentAs() 53  
 dw.saveDocumentAsTemplate() 54  
 dw.saveFrameset() 54  
 dw.saveFramesetAs() 54  
 dw.selectAll() 108  
 dw.setActiveWindow() 152  
 dw.setFloaterVisibility() 152  
 dw.setHideAllFloaters() 141  
 dw.setShowInvisibleElements() 141  
 dw.setShowStatusBar() 141  
 dw.setUpComplexFind() 56  
 dw.setUpComplexFindReplace() 57  
 dw.setUpFind() 58  
 dw.setUpFindReplace() 59  
 dw.showAboutBox() 73  
 dw.showFindDialog() 59  
 dw.showFindReplaceDialog() 60  
 dw.showGridSettingsDialog() 149  
 dw.showPreferencesDialog() 74  
 dw.showProperties() 152  
 dw.showQuickTagEditor() 104  
 dw.startRecording() 77  
 dw.stopRecording() 78  
 dw.templatePalette.deleteSelectedTemplate() 98  
 dw.templatePalette.getSelectedTemplate() 98  
 dw.templatePalette.newBlankTemplate() 99  
 dw.templatePalette.renameSelectedTemplate() 99  
 dw.timelineInspector.addBehavior() 130  
 dw.timelineInspector.addFrame() 130  
 dw.timelineInspector.addKeyframe() 130  
 dw.timelineInspector.addObject() 131  
 dw.timelineInspector.addTimeline() 131  
 dw.timelineInspector.canAddFrame() 169  
 dw.timelineInspector.canAddKeyFrame() 169  
 dw.timelineInspector.canChangeObject() 170  
 dw.timelineInspector.canRemoveFrame() 170  
 dw.timelineInspector.canRemoveKeyFrame() 170  
 dw.timelineInspector.canRemoveObject() 170  
 dw.timelineInspector.changeObject() 131  
 dw.timelineInspector.getAutoplay() 131  
 dw.timelineInspector.getCurrentFrame() 132  
 dw.timelineInspector.getLoop() 132

- dw.timelineInspector.recordPathOfLayer() 132
- dw.timelineInspector.removeBehavior() 132
- dw.timelineInspector.removeFrame() 133
- dw.timelineInspector.removeKeyframe() 133
- dw.timelineInspector.removeObject() 133
- dw.timelineInspector.removeTimeline() 133
- dw.timelineInspector.renameTimeline() 134
- dw.timelineInspector.setAutoplay() 134
- dw.timelineInspector.setCurrentFrame() 134
- dw.timelineInspector.setLoop() 134
- dw.toggleFloater() 153
- dw.undo() 78
- dw.updatePages() 97
- dw.useTranslatedSource() 145
- dynamic menus
  - sample code 242
  - user experience 236

## E

- editAttribute() 62
- editCommandList() 37
- editFontList() 71
- editLockedRegions() 144
- editSelectedStyle()
  - dreamweaver.cssStylePalette.editSelectedStyle() 41
  - dreamweaver.htmlStylePalette.editSelectedStyle() 83
- editStyleSheet() 41
- element node 15
- enablers
  - return value 158
  - using 20
- endOfDocument() 87
- endOfLine() 87
- escape() 10
- event handlers
  - in behavior dialog boxes 262
  - in extension files 18
  - returning a value from 263
- events
  - in extension files 10
- execJsInFireworks() 194
- exists() 179
- exitBlock() 62
- exportCSS() 49
- exportEditableRegionsAsXML() 50
- external JavaScript files 18

## F

- file (field) object 10
- file i/o 178
- FilePathToLocalURL() 192
- filePathToLocalURL() 187
- files on disk
  - copying 178
  - creating (HTML files) 49
  - creating (non-HTML files) 182
  - reading 181
  - removing 182
  - writing to 182
- findLinkSource() 112
- findNext() 55
- Fireworks
  - integration example 198
- floating palettes
  - API 255
  - performance issues 257
  - sample code 259
  - user experience 253
- focus() 10
- form object 10
- function object 10
- FWLaunch.bringDWToFront() 193
- FWLaunch.bringFWToFront() 193
- FWLaunch.execJsInFireworks() 194
- FWLaunch.getJsResponse() 194
- FWLaunch.mayLaunchFireworks() 196
- FWLaunch.optimizeInFireworks() 196
- FWLaunch.validateFireworks() 197

## G

- get()
  - MMNotes.get() 186
  - site.get() 113
- getActiveWindow() 150
- getAttachedTemplate() 94
- getAttribute() 15
- getAttributes() 179
- getAutoplay() 131
- getBehavior() 23
- getBehaviorAt() 28
- getBehaviorCount() 28
- getBehaviorElement() 25
- getBehaviorEvent() 153
- getBehaviorTag() 26



- getBrowserList() 45
- getCheckOutUser() 113
- getCheckOutUserForFile() 114
- getClipboardText() 36
- getConfigurationPath() 101
- getConnectionState() 114
- getCurrentFrame() 132
- getCurrentSite() 114
- getDocumentDOM()
  - dreamweaver.getDocumentDOM() 21
  - importance of 21
- getDocumentList() 151
- getDocumentPath() 101
- getDynamicContent() 238
- getEditableRegionList() 94
- getEditableRetionList() 95
- getEditNoFramesContent() 135
- getElementRef() 75
- getElementsByTagName()
  - for document objects 14
  - for tag objects 15
- getExtensionEditorList() 45
- getFile() 203
- getFileCallback() 205
- getFloaterVisibility() 151
- getFocus()
  - dom.getFocus() 150
  - dreamweaver.getFocus() 151
  - site.getFocus() 115
- getFontList() 71
- getFontMarkup() 62
- getFrameNames() 60
- getHideAllFloaters() 140
- getIsLibraryDocument() 94
- getIsTemplateDocument() 95
- getJsResponse() 194
- getKeyCount() 186
- getKeys() 186
- getKeyState() 72
- getLinkHref() 62
- getLinkTarget() 63
- getLinkVisibility() 115
- getListTag() 63
- getLoop() 132
- getMenuNeedsUpdating() 99
- getModificationDate() 180
- GetNote() 190

- GetNoteLength() 189
- GetNotesKeyCount() 190
- GetNotesKeys() 191
- getObjectRefs() 154
- getObjectTags() 155
- getPreventLayerOverlaps() 135
- getPrimaryBrowser() 45
- getPrimaryExtensionEditor() 46
- getRecentFileList() 50
- getRedoText() 76
- getRulerOrigin() 146
- getRulerUnits() 146
- getSecondaryBrowser() 46
- getSelectedBehavior() 29
- getSelectedEditableRegion() 95
- getSelectedItem() 97
- getSelectedNode() 105
- getSelectedStyle()
  - dreamweaver.cssStylePalette.getSelectedStyle() 41
  - dreamweaver.htmlStylePalette.getSelectedStyle() 83
- getSelectedTarget() 42
- getSelectedTemplate() 98
- getSelection()
  - dom.getSelection() 105
  - dreamweaver.getSelection() 156
  - site.getSelection() 115
- getShowDependents() 142
- getShowDialogsOnInsert() 73
- getShowFrameBorders() 135
- getShowGrid() 135
- getShowHeaderView() 136
- getShowHiddenFiles() 142
- getShowImageMaps() 136
- getShowInvisibleElements() 140
- getShowLayerBorders() 136
- getShowPageTitles() 142
- getShowRulers() 136
- getShowStatusBar() 141
- getShowTableBorders() 137
- getShowToolTips() 142
- getShowTracingImage() 137
- getSiteRoot() 102
- GetSiteRootForFile() 191
- getSiteRootForFile() 187
- getSites() 115
- getSnapToGrid() 137
- getStepCount() 79

- getStepsAsJavaScript() 80
- getStyles()
  - dreamweaver.cssStylePalette.getStyles() 43
  - dreamweaver.htmlStylePalette.getStyles() 83
- getTableExtent() 126
- getText() 206
- getTextAlignment() 63
- getTextCallback() 206
- getTextFormat() 63
- getTokens() 123
- getTracingImageOpacity() 146
- getTranslatedAttribute() 15
- getTranslatorInfo() 275
- getTranslatorList() 145
- getUndoState() 80
- getUndoText() 77
- GetVersionName() 192
- getVersionName() 187
- GetVersionNum() 192
- getVersionNum() 187
- getWindowTitle() 150

## H

- hasCharacterMarkup() 64
- hasChildNodes()
  - for comment objects 17
  - for document objects 14
  - for tag objects 15
  - for text objects 17
- hasTracingImage() 164
- hasTranslatedAttributes() 15
- helper functions, in behaviors 263
- hidden (field) object 10
- hotspot functions 91

## I

- identifyBehaviorArguments() 267
- image (field) object 10
- image map functions 91
- image object 10
- importXMLIntoTemplate() 50
- increaseColspan() 127
- increaseRowspan() 127
- indent() 64
- InfoPrefs 191
- initialPosition() 256

- initialTabs() 257
- innerHTML property 15
- insertHTML() 65
- insertLibraryItem() 95
- insertObject() 65
- insertTableColumns() 127
- insertTableRows() 127
- insertText() 66
- inspectBehavior() 269
- inspectSelection() 248
- invertSelection() 116
- isCommandChecked() 238
- isRecording() 169
- item() 10

## J

- JavaScript
  - books 8
  - syntax 8
- javascript URLs 18
- JS\_BooleanToValue() 215
- JS\_DefineFunction() 212
- JS\_DoubleToValue() 215
- JS\_ExecuteScript() 217
- JS\_GetArrayLength() 216
- JS\_GetElement() 216
- JS\_IntegerToValue() 215
- JS\_NewArrayObject() 216
- JS\_ObjectToValue() 215
- JS\_ObjectType() 215
- JS\_ReportError() 218
- JS\_SetElement() 217
- JS\_StringToValue() 214
- JS\_ValueToBoolean() 214
- JS\_ValueToDouble() 213
- JS\_ValueToInteger() 213
- JS\_ValueToObject() 214
- JS\_ValueToString() 213
- JSBool 212
- JSContext 211
- JSNative 212
- JSObject 211
- jsval 212

## L

- language information 13
- latin1ToNative() 124
- layer object 10
- listFolder() 181
- loadTracingImage() 147
- LocalURLToFilePath() 192
- localURLToFilePath() 188
- locateInSite() 112
- locked content, inspecting 289
- \*LOCKED\* keyword 289

## M

- makeEditable() 116
- makeNewDreamweaverFile() 116
- makeNewFolder() 116
- makeSizesEqual() 91
- markSelectionAsEditable() 96
- math object 10
- mayLaunchFireworks() 196
- menu commands
  - API 237
  - sample code 240
  - user experience 236
- mergeTableCells() 128
- mm\_jsapi.h
  - including 211
  - sample file 218
- MM\_returnValue 263
- MMHttp.clearTemp() 203
- MMHttp.getFile() 203
- MMHttp.getFileCallback() 205
- MMHttp.getText() 206
- MMHttp.getTextCallback() 206
- MMHttp.postText() 207
- MMHttp.postTextCallback() 207
- MMInfo.h 188
- MMNotes object 184
- MMNotes.close() 185
- MMNotes.filePathToLocalURL() 187
- MMNotes.get() 186
- MMNotes.getKeyCount() 186
- MMNotes.getKeys() 186
- MMNotes.getSiteRootForFile() 187
- MMNotes.getVersionName() 187
- MMNotes.getVersionNum() 187
- MMNotes.localURLToFilePath() 188

- MMNotes.open() 184
- MMNotes.remove() 185
- MMNotes.set() 185
- moveBehaviorDown() 30
- moveBehaviorUp() 31
- moveSelectionBy() 92

## N

- nativeToLatin1() 124
- navigator object 10
- newBlankTemplate() 99
- newBlock() 66
- newEditableRegion() 96
- newFromDocument() 97
- newFromTemplate() 50
- newHomePage() 117
- newSite() 117
- newStyle()
  - dreamweaver.cssStylePalette.newStyle() 43
  - dreamweaver.htmlStylePalette.newStyle() 84
- nextParagraph() 88
- nextWord() 88
- node constants 14
- Node.COMMENT\_NODE 14
- Node.DOCUMENT\_NODE 14
- Node.ELEMENT\_NODE 14
- odelist object 10
- nodes 14
- Node.TEXT\_NODE 14
- nodeToOffsets()
  - dom.nodeToOffsets() 106
  - dreamweaver.nodeToOffsets() 156
- nodeType property
  - of comment objects 17
  - of document objects 14
  - of tag objects 15
  - of text objects 17
- \_notes folder 184
- notifyMenuUpdated() 100
- number object 10

## O

- object object 10
- objects
  - adding to Insert menu 225
  - adding to Object palette 224
  - API 222
  - user experience 222
- objectTag() 223
- offsetsToNode()
  - dom.offsetsToNode() 106
  - dreamweaver.offsetsToNode() 157
- onBlur 10
- onChange 10
- onClick 10
- onFocus 10
- onLoad 10
- onMouseDown 10
- onMouseOut 10
- onMouseOver 10
- onMouseUp 10
- onResize 10
- open()
  - MMNotes.open() 184
  - site.open() 117
- openDocument() 51
- openDocumentFromSite() 51
- openInFrame() 52
- OpenNotesFile() 188
- openWithApp() 46
- openWithBrowseDialog() 47
- openWithExternalTextEditor() 47
- openWithImageEditor() 47
- optimizeInFireworks() 196
- option object 10
- outdent() 67
- outerHTML property 15

## P

- pageDown() 88
- pageUp() 89
- parentNode property
  - of comment objects 17
  - of document objects 14
  - of tag objects 15
  - of text objects 17
- parentWindow property 14
- password (field) object 10

- playAllPlugins() 147
- playPlugin() 147
- playRecordedCommand() 77
- popupAction() 27
- popupCommand() 157
- postText() 207
- postTextCallback() 207
- previousParagraph() 89
- previousWord() 89
- property inspectors
  - \*LOCKED\* keyword 289
  - API 247
  - for locked content 289
  - lightning bolt icon 283
  - required functions 247
  - sample code 249
  - translated attributes in 283
  - user experience 246
- put() 118

## Q

- quitApplication() 73

## R

- radio object 10
- read() 181
- reapplyBehaviors() 23
- receiveArguments()
  - in menu commands 238
  - in regular commands 230
- recordPathOfLayer() 132
- recreateCache() 118
- recreateFromDocument() 98
- redo()
  - dom.redo() 76
  - dreamweaver.redo() 77
- refresh() 118
- regexp object 10
- relativeToAbsoluteURL() 102
- releaseDocument() 52
- reloadMenus() 100
- remoteIsValid() 119
- remove() 182, 185
- removeAllTableHeights() 128
- removeAllTableWidths() 128
- removeAttribute() 15

- removeBehavior()
  - dom.removeBehavior() 24
  - dreamweaver.timelineInspector.removeBehavior() 132
- removeCharacterMarkup() 67
- removeCSSStyle() 40
- removeEditableRegion() 96
- removeFontMarkup() 67
- removeFrame() 133
- removeKeyframe() 133
- removeLink()
  - dom.removeLink() 67
  - site.removeLink() 119
- RemoveNote() 189
- removeObject() 133
- removeTimeline() 133
- renameSelectedItem() 98
- renameSelectedTemplate() 99
- renameSelection() 119
- renameTimeline() 134
- replace() 55
- replaceAll() 55
- replaySteps() 81
- reset object 10
- resizeSelection() 68
- resizeSelectionBy() 92
- resizeTo() 10
- revertDocument() 52
- runCommand() 37
- runTranslator() 144

**S**

- saveAll() 53
- saveAsCommand() 81
- saveAsImage() 119
- saveDocument() 53
- saveDocumentAs() 53
- saveDocumentAsTemplate() 54
- saveFrameset() 54
- saveFramesetAs() 54
- select object 10
- select() 10
- selectAll()
  - dom.selectAll() 107
  - dreamweaver.selectAll() 108
  - site.selectAll() 120
- selectChild() 103
- selectHomePage() 120
- selectionChanged() 256
- selectNewer() 120
- selectParent() 103
- selectTable() 107
- set() 185
- setActiveWindow() 152
- setAsHomePage() 120
- setAttribute() 15
- setAttributeWithErrorChecking() 68
- setAutoplay() 134
- setConnectionState() 121
- setCurrentFrame() 134
- setCurrentSite() 121
- setEditNoFramesContent() 137
- setFloaterVisibility() 152
- setFocus() 121
- setHideAllFloaters() 141
- setInterval() 10
- setLayerTag() 93
- setLayout() 121
- setLinkHref() 68
- setLinkTarget() 69
- setLinkVisibility() 122
- setListBoxKind() 69
- setListTag() 70
- setLoop() 134
- setMenuText() 239
- SetNote() 189
- setPreventLayerOverlaps() 138
- setRulerOrigin() 147
- setRulerUnits() 148
- setSelectedBehavior() 32
- setSelectedNode() 107
- setSelection()
  - dom.setSelection() 108
  - dreamweaver.setSelection() 158
  - site.setSelection() 122
- setShowDependents() 143
- setShowFrameBorders() 138
- setShowGrid() 138
- setShowHeaderView() 138
- setShowHiddenFiles() 143
- setShowImageMaps() 139
- setShowInvisibleElements() 141
- setShowLayerBorders() 139
- setShowPageTitles() 143
- setShowRulers() 139

- setShowStatusBar() 141
- setShowTableBorders() 139
- setShowToolTips() 143
- setShowTracingImage() 140
- setSnapToGrid() 140
- setTableCellTag() 128
- setTableColumns() 129
- setTableRows() 129
- setTextAlignment() 70
- setTextFieldKind() 70
- setTimeout() 10
  - in floating palettes 257
  - use with FWLaunch 198
- setTracingImageOpacity() 148
- setTracingImagePosition() 148
- setUndoState() 82
- setUpComplexFind() 56
- setUpComplexFindReplace() 57
- setUpFind() 58
- setUpFindReplace() 59
- showAboutBox() 73
- showFindDialog() 59
- showFindReplaceDialog() 60
- showFontColorDialog() 70
- showGridSettingsDialog() 149
- showInsertTableRowsOrColumnsDialog() 129
- showListPropertiesDialog() 69
- showPagePropertiesDialog() 75
- showPreferencesDialog() 74
- showProperties() 152
- showQuickTagEditor() 104
- shutdown commands 18
- Shutdown folder 18
- site object
  - methods of 20
  - properties of 13
- site.addLinkToExistingFile() 109
- site.addLinkToNewFile() 109
- site.canAddLinkToFile() 171
- site.canChangeLink() 171
- site.canCheckIn() 171
- site.canCheckOut() 172
- site.canConnect() 172
- site.canFindLinkSource() 172
- site.canGet() 173
- site.canLocateInSite() 173
- site.canMakeEditable() 173
- site.canMakeNewFileOrFolder() 174
- site.canOpen() 174
- site.canPut() 174
- site.canRecreateCache() 174
- site.canRefresh() 175
- site.canRemoveLink() 175
- site.canSelectNewer() 175
- site.canSetLayout() 175
- site.canSynchronize() 176
- site.canUndoCheckOut() 176
- site.canViewAsRoot() 176
- site.changeLink() 110
- site.changeLinkSitewide() 109
- site.checkIn() 110
- site.checkLinks() 110
- site.checkOut() 111
- site.checkTargetBrowsers() 111
- site.defineSites() 111
- site.deleteSelection() 112
- site.findLinkSource() 112
- site.get() 113
- site.getCheckOutUser() 113
- site.getCheckOutUserForFile() 114
- site.getConnectionState() 114
- site.getCurrentSite() 114
- site.getFocus() 115
- site.getLinkVisibility() 115
- site.getSelection() 115
- site.getShowDependents() 142
- site.getShowHiddenFiles() 142
- site.getShowPageTitles() 142
- site.getShowToolTips() 142
- site.getSites() 115
- site.invertSelection() 116
- site.locateInSite() 112
- site.makeEditable() 116
- site.makeNewDreamweaverFile() 116
- site.makeNewFolder() 116
- site.newHomePage() 117
- site.newSite() 117
- site.open() 117
- site.put() 118
- site.recreateCache() 118
- site.refresh() 118
- site.remoteIsValid() 119
- site.removeLink() 119
- site.renameSelection() 119

- site.saveAsImage() 119
- site.selectAll() 120
- site.selectHomePage() 120
- site.selectNewer() 120
- site.setAsHomePage() 120
- site.setConnectionState() 121
- site.setCurrentSite() 121
- site.setFocus() 121
- site.setLayout() 121
- site.setLinkVisibility() 122
- site.setSelection() 122
- site.setShowDependents() 143
- site.setShowHiddenFiles() 143
- site.setShowPageTitles() 143
- site.setShowToolTips() 143
- site.synchronize() 122
- site.undoCheckOut() 122
- site.viewAsRoot() 123
- snapTracingImageToSelection() 148
- splitFrame() 60
- splitTableCell() 129
- startOfDocument() 90
- startOfLine() 90
- startRecording() 77
- startup commands 18
- Startup folder 18
- status codes 202
- statusCode property 202
- stopAllPlugins() 149
- stopPlugin() 149
- stopRecording() 78
- string object 10
- stripTag() 103
- submit object 10
- synchronize() 122

## T

- tag object 15
- tagName property 15
- text (field) object 10
- text node 17
- text object 17
- textarea object 10
- toggleFloater() 153

- translated attributes
  - finding in tags 15
  - individual 278
  - inspecting 283
  - multiple 279
- translated tags, inspecting 289
- translateMarkup() 277
- translators
  - attribute 278
  - block/tag 284
  - debugging 293
- typeof operator 177

## U

- undo()
  - dom.undo() 76
  - dreamweaver.undo() 78
- undoCheckOut() 122
- unescape() 10
- updateCurrentPage() 96
- updatePages() 97
- URL property 14
- useTranslatedSource() 145

## V

- validateFireworks() 197
- versioning 13
- viewAsRoot() 123

## W

- W3C 10
- window object 10
- windowDimensions()
  - in behavior actions 270
  - in menu commands 239
  - in object files 224
  - in regular commands 231
- wrapTag() 104
- write() 182

